VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Urychlení rozpoznávání detekce obličejů v obraze s využitím GPU
Acceleration of Face Detection by Making Use of GPU Computing

2012 Bc. Jaromír Krpec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jaromír Krpec**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Urychlení rozpoznávání detekce obličejů v obraze s využitím GPU
Acceleration of Face Detection by Making Use of GPU Computing

Zásady pro vypracování:

Detekce obličejů v obraze patří dnes k velmi využívaným algoritmům a využívá se například v oblastech bezpečnostních aplikací. Cílem této práce je přinést souhrny přehled základních metod detekce a zaměřit se na možnosti urychlení těchto algoritmů s využitím GPU.

1. Seznamte se s algoritmy a postupy pro problematikou rozpoznávání a detekce obličejů v obraze. Teoretický rozbor krátce shrňte ve své práci. Dále se seznamte s možnostmi urychlení výpočtu aplikací s využitím GPU. Zaměřte se na technologii CUDA.
2. Naimplementujte a otestujte aplikaci umožňující porovnat rychlost a kvalitu detekce pro metodu Viola & Jones. Dále srovnejte výpočet jak pro aplikaci využívající CPU tak aplikaci s urychlením výpočtu při využití GPU. Srovnejte a popište možnosti, ve kterých lze GPU využít pro urychlení aplikace.
3. Ukázkové aplikace jednotlivých metod detekce porovnejte na různých vstupech dat (rozměry, počet proměnných).

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Němec, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 27. Dubna 2012 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal

V Ostravě 27. Dubna 2012 .....

Rád bych poděkoval vedoucímu práce Ing. Martinu Němcovi, Ph.D. za podnětné připomínky a cenné rady při vytváření praktické i teoretické části této diplomové práce.

Abstrakt
Detekce obličejů je velmi užitečným a důležitým prvkem, jenž zasahuje do mnoha zcela odlišných oborů. Jelikož se jedná o časově náročný problém, cílem této práce je především zjistit, zda je výhodné použít pro proces detekce postup, kdy by byl výpočet prováděn pomocí grafické karty.
K ověření byl vybrán algoritmus Viola a Jones, který byl nejprve implementován v základní verzi pro procesor. Poté byla aplikace upravena tak, aby byl využit výpočet pomocí grafické karty. Nakonec jsou jednotlivé programy porovnány.

Klířová slova: CUDA, GPU, Detekce obličejů, Viola a Jones algoritmus

Abstract
Face detection is very useful and important for many different disciplines. On the other side, because it is very time-consuming problem, the target of this work is to determine, whether it is advantageous to use the graphic card for detection process. The Viola and Jones algorithm was chosen for the classic implementation using the processors. Then, the program was modified to the graphic card performed the main detection process. At the end, final applications are compared and the results are presented in this work.

Keywords: CUDA, GPU, Face Detection, Viola and Jones algorithm

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojových kódů

1. Úvod

Detekování obličejů v obraze je poměrně složitý i časově náročný problém, který si však našel upotřebení napříč různými obory, ať už se jedná například o reklamu, robotiku, bezpečnost či běžně dostupnou technologii u digitálních fotoaparátů. Právě díky různorodým využitím a zároveň s neustále se zvyšující výkonnosti počítačů, obor zpracování obrazu zažívá velký rozvoj a s tím i samotná detekce objektů v obraze.

Přesto však i na dnešním běžném hardwaru je proces detekování objektů, nejen tedy detekce obličejů, velmi náročný. A to zejména v případě, kdy jsou zpracovávána větší vstupní obrazová data. Totéž platí i pro detekování objektů v obraze, které jsou snímány kamerou. Zejména z těchto důvodů je vhodné detekční proceduru urychlit tak, aby ji bylo možné používat především pro nasazení do aplikací zpracující obraz v reálném čase.

Jelikož v posledních letech nastal velký rozvoj grafických karet, jejichž výkon dokázal značně předběhnout procesory běžných počítačů, byla zvolena akcelerace detekčního procesu právě prostřednictvím využití GPU. Ty dnes díky novým technologiím mohou být použity i k náročným paralelním výpočtům, které nesouvisí jen s vykreslováním 2D či 3D grafiky, ale mohou se týkat libovolných výpočtů, které je možné zpracovávat souběžně.

Během této práce byly připraveny různé verze programu využívající ke svému výpočtu, jak CPU tak i GPU, které byly postupně otestovány především z hlediska výkonu. Obě verze budou následně navzájem porovnány z hlediska rychlosti detekce.

Vypracovaná práce je rozdělena do sedmi kapitol. První kapitola se obecně zabývá různými metodami, jež lze využít pro detekci obličejů. Principy některých vybraných metod jsou zde také zevrubně popsány, aby byly zřejmé principy jejich fungování a rozdíl oproti implementované metodě. Druhá kapitola se zabývá vybranou technikou Viola a Jones, patřící do skupiny algoritmů strojového učení, která byla následně také implementována a porovnávána. Algoritmus je zde teoreticky popsán z hlediska trénování a detekce. Čtvrtá a pátá kapitola se zabývá samotnou implementací učitele a detektoru na základě teoretických znalostí z třetí kapitoly, kde jsou vysvětleny nejen jedno-vláknové, ale také více-vláknové varianty. Šestá kapitola je pak celá věnována technologii CUDA a postupu při implementaci detektoru pomocí této technologie. Poslední sedmá kapitola se pak zaobírá samotným testováním a porovnáním jednotlivých verzí detektorů. Uvedené výsledky nejsou prezentovány jen z hlediska výkonu samotného detektoru, ale jsou zde uvedeny také náročnosti dílčích kroků jednotlivých kroků prováděných během celého procesu detekce.

2. Vybrané metody detekce obličejů

Metody pro detekci obličejů ve vstupním obraze je možné obecně rozdělit do několika hlavních skupin podle přístupu, jakým k detekci přistupují. Dle [2] je možné rozdělit algoritmy do čtyř hlavních kategorií.

Znalostní metody	Multirezoluční metoda založená na pravidlech
Metody založené na invariantních rysech	Textury
	Barva kůže
	Násobné rysy
Metody porovnávání se šablonou	Předdefinované šablony
	Deformující šablony
Metody založené na vzhledu	Eigenface
	Distribuční metody
	Neuronové sítě
	Support Vector Machine
	Hidden Markov Model
	Boost

Tabulka 2-1 Rozdělení metod detekce obličejů

- **Znalostní metody (Knowledge-based methods)**
  Znalostní metoda využívá informací o lidské tváři, resp. z jakých částí se tvář skládá, kde se nachází a v jakém jsou jednotlivé části vztahu. Detekce probíhá tak, že se nejprve naleznou části lidské tváře a následně se hledají vzájemné vztahy, načež se rozhodne, zda se jedná o tvář, či nikoliv.

- **Metody založené na invariantních rysech (Feature invariant approaches)**
  Využívají se neměnné rysy tváře nepodléhající změně osvětlení či natočení. Také u této metody dochází k detekování obličejových rysů a na základě vztahů mezi nimi se rozhoduje, zda se může jednat o tvář.

- **Metody porovnávání se šablonou (Template matching methods)**
  Metoda porovnávání výseku obrazu s předpřipravenou šablonou, které mohou obsahovat celý obličej, nebo jen jeho části. Na hodnotě výsledné korelace vstupního obrazu a vzoru se rozhodne, zda je daná část tváří.

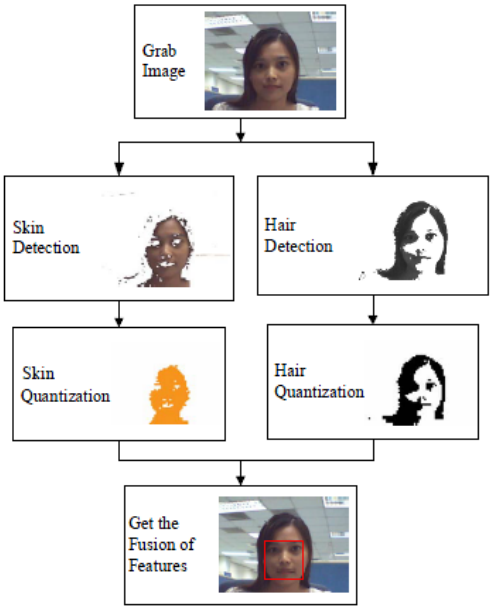
- **Metody založené na vzhledu (Appearance-based methods)**
  Vzory, jež jsou základním prvkem této metody, jsou naučeny pomocí sady obrazů určených pro trénování. Získané předlohy jsou poté použity pro vlastní detekci obličejů v obraze.

2.1 Metoda založená na barvě kůže

Metoda vyhledávající v detekovaném obraze oblasti, které mají podobnost s barvou kůže, přičemž se většinou využívá barevný prostor YC_bC_r. Tento prostor je složen ze tří složek, nicméně informaci o barvě nesou pouze dvě. Prvek C_b určuje rozdíl mezi modrou složkou a referenční hodnotou, C_r je rozdíl mezi červenou složkou a referenční hodnotou. Třetí prvek Y vyjadřuje luminanci, čili sílu jasu. Vstupní obraz zadaný v obvyklém prostoru RGB je možné snadno přepočtem převést do prostoru YC_bC_r

$$Y = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B\,,$$
$$C_b = -0{,}1687 \cdot R - 0{,}3313 \cdot G + 0{,}5 \cdot B + 128\,,$$
$$C_r = 0{,}5 \cdot R - 0{,}4187 \cdot G - 0{,}0813 \cdot B + 128\,.$$

Nalezené oblasti obsahující barevné body, které odpovídají definovanému rozsahu odpovídajícímu barvě kůže, jsou ohraničovány a spojovány. Zároveň je nutné aplikovat určitá pravidla týkající se tvarů oblastí, které vyloučí falešně označené oblasti, které mohou patřit například jiným částem lidského těla.



Obrázek 2-1 Schéma detekování obličeje pomocí barvy kůže a vlasů [3]

Na obrázku 2-1 je možné vidět rozšíření detekce barvy kůže o detekování oblasti vlasů [3], které má za následek snížení falešně označených obličejů. Vybrané oblasti popisující kůži a vlasy jsou během zpracování detektorem následně kvantovány použitím čtverců o rozměrech 5x5 pixelů.
V posledním kroku se zjišťuje, zda oblasti kůže a vlasů spolu sousedí. Úspěšnost tohoto algoritmu pro detekci jedné tváře ve zpracovávaném obrazu podle autorů tohoto postupu dosahuje až 92%.

2.2 Template matching

Metoda Template matching spočívá v porovnávání vybrané části vstupního obrazu s předzpracovanými obličeji. Pro jednotlivé části obrazu se určuje korelace mezi ní a tváří na šabloně a na základě její velikosti se nakonec určí, zda vybraná část obsahuje či neobsahuje obličej.

Prvním krokem je tedy vytvoření šablon, které jsou následně využity pro detekci. Tato předloha se nazývá průměrný obličej a je vytvořen z několika různých obličejů:

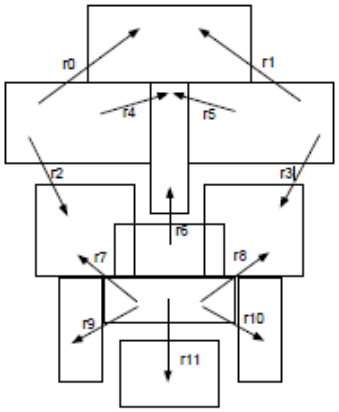
Obrázek 2-2 Vlevo průměrný obličej a vpravo vytvořená šablona k porovnávání [2]

Vytvořenou šablonu je při detekci nutné také při průchodu zvětšovat, aby bylo možné detekovat i tváře, které jsou větší než připravená šablona. Stejně tak je možné se šablonu provádět rotace. Template matching je vhodné používat zejména pro detekci tváří z frontálního pohledu [2], avšak za použití tzv. deformovatelných šablon, které se snaží přizpůsobit předpřipravené šablony podle tváře ve vstupním obrazu, je možné používat tuto metodu i na lokalizaci obličejů v různých pozicích.

Template matching lze kombinovat s detekcí barvy kůže [4]. Podle tohoto postupu jsou v prvním kroku ve vstupním obrazu nalezena místa, která obsahují barevné body odpovídající barvě

kůže. Všechny vybrané oblasti, které svým tvarem mohou být pokládány za tváře, jsou následně porovnány s šablonou s přizpůsobenou velikost dané oblasti.

Není však nutné vždy porovnávat celé šablony obličejů. Metoda Ratio-Template předpokládá, že obličej je možné rozdělit do několika menších částí [5]. Pro každou z nich existuje šablona popisující oblast, kterou v dané části obrazu očekáváme a právě s ní dochází k porovnávání.



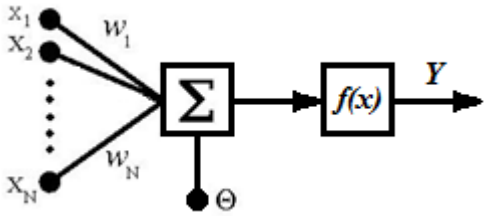
Obrázek 2-3 Oblasti tváře [5]

Navíc, jak ukazuje obrázek 2-3, jsou mezi jednotlivými oblastmi určité vztahy. A to nejen sousednost jednotlivých částí, ale také určitá závislost jasů. Hodnoty jasů mezi určitými oblastmi jsou totiž nezávislé na světelných podmínkách. Autor uvádí, že tento postup dosahuje 80% úspěšností a jen malého množství false positives.

2.3 Neuronové sítě

Neuronové stě jsou určitým zpodobněním fungování lidského mozku, kdy umělé neurony používány v neuronových sítích odpovídají neuronům, jež se nacházejí právě v lidském mozku. Jednotlivé neurony jsou v síti propojeny, předávají si postupně určité informace, které se postupně přeměňují, až se dostanou na výstup.

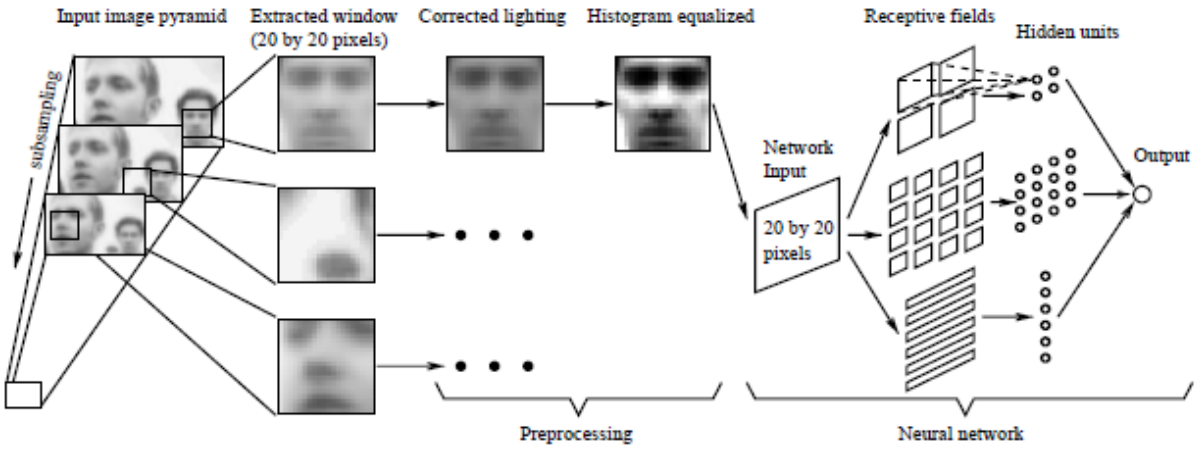
Základem neuronových sítí je tedy neuron, který obsahuje *n* vstupů, ale pouze jeden jediný výstup, který je možné využít jako vstup do dalších neuronů.



Obrázek 2-4 Umělý neuron

Obrázek 2-4 znázorňuje umělý neuron se vstupy $x_1, x_2, \dots x_N$, váhy vstupů $w_1, w_2, \dots w_N$, přenosovou funkcí $f(x)$, prahem θ a výstupem Y. Právě v neuronu se vstupní informace transformuje a vzájemná propojení těchto základních prvků vytváří neuronovou síť, neboť samotný neuron není schopen vykonat příliš složitou funkci.

Metody detekce obličejů pomocí neuronových sítí je nutné nejdříve natrénovat, čili nastavit hodnoty vah w_i, které vyjadřují zkušenost daného neuronu. Čím je hodnota vyšší, tím je příslušný vstup důležitější. Proces trénování trvá tak dlouho, dokud není dosaženo přednastavených výsledků. Jakmile je trénování jednou ukončeno, může již proběhnout detekce na zcela neznámých vstupech.



Obrázek 2-5 Algoritmus detekce neuronovými sítěmi [14]

Stejně tak lze neuronových sítí možné použít pro detekci jednotlivých rysů ve vybraném detekčním okně. Podle [14] je také možné použít zároveň detekci pomocí více neuronových sítí, které jsou různě natrénovány za účelem odstranění špatně označených obrazů. Právě obrázku 2-5 je znázorněn algoritmus, který používá tři skryté vrstvy. První vrstva rozeznává významnější rysy tváře, jako jsou oka, nos či koutky úst. Druhá vrstva hledá jemnější rysy a třetí pak ústa a oči. Výstupem je hodnota 1 nebo -1 říkající, zda se v aktuálně zpracovaném okně nachází obličej nebo nikoliv.

2.4 Eigenface

Jelikož tváře jsou si ve významnějších rysech podobné, hlavní myšlenkou metody Eigenface je nalézt příslušné vektory, které by co možná nejlépe dokázaly popsat rozložení obrázků obsahující tvář v celém prostoru obrázků. Příslušné vektory pak ohraničují prostor tváří. Následná detekce pak probíhá tak, že pro vybrané okno je určena vzdálenost od prostoru obličejů a na základě této hodnoty se určí, zda se v okně vyskytuje tvář.

Před detekcí je nutné nejprve určit příslušný prostor. Máme k dispozici M obrázků se stejnými rozměry $N \times N$, ze kterých je nejprve vytvořen vektor

$$I_i = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & & a_{2N} \\ \vdots & & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix} \longrightarrow \begin{bmatrix} a_{11} \\ \vdots \\ a_{1N} \\ \vdots \\ a_{NN} \end{bmatrix} = \Gamma_i .$$

Účelem celé metody je nalézt takové vektory, které co možná nejlépe popíší obrázky tváří vstupní množiny. Získané vektory jsou vlastními vektory matice. Výpočet dále pokračuje v následujících krocích

1. Je dána množina obrázku označená jako $I_1, I_2, \ldots I_M$,

Obrázek 2-6 Sada vstupních obrázků [13]

2. Reprezentovat každý obraz jako vektor Γ_i
3. Vypočte se průměrný obrázek ze zadané množiny

$$\psi = \frac{1}{M} \cdot \sum_{i=1}^{M} \Gamma_i .$$

Obrázek 2-7 Průměrný obličej [13]

4. Z každého vstupního obrazu je odečten průměrný obraz
$$\phi_i = \Gamma_i - \psi .$$
5. Získané vektory jsou poté vloženy do matice *A*, ve které každý řádek i představuje odpovídající vektor ϕ_i
6. Kovarianční matici s rozměry $n \times n$ získáme jako
$$C = \frac{1}{M}\sum_{n=1}^{M} \phi_n \phi_n^T = AA^T \text{ , kde } A = [\phi_1, \phi_2, \dots \phi_M] .$$
Jak je vidět, v tomto kroku vznikne matice velikosti $N^2 \times N^2$, což pro další operaci vzhledem k této velikosti není příliš vhodné. Právě proto se předchozí výpočet kovarianční matice obvykle provádí takto
$$C' = A^T A .$$
Vzniklá matice má rozměry $M \times M$.
7. Poté jsou vypočteny vlastní čísla λ a vlastní vektory *u* matice C. Nicméně z předešlého bodu vyplývá, že se používá matice C'. Budou tedy získávána vlastní čísla λ a vlastní vektory v. Musí být vyřešena následující rovnice
$$C'v = \lambda v .$$
Je možné toto takto provést, jelikož matice C a C' mají stejné vlastní čísla a vztah mezi vlastními vektory, kdy $u = Av$
$$C'v = \lambda v ,$$
$$A^T Av = \lambda v ,$$
$$AA^T v = \lambda Av ,$$
$$CAv = \lambda Av .$$
8. Získané vektory jsou normalizovány
$$v = \frac{v}{|v|} .$$
Pro velikost vektoru platí
$$|v| = \sqrt{\sum v_i^2} .$$
9. Vybere se jen *K* vlastních vektorů odpovídající největším hodnotám vlastních čísel

Obrázek 2-8 Výsledné eigenface [13]

Odečteme-li od každého obrazu v tréninkové množině příslušný vektor ϕ_i, pak tento obličej může být reprezentován jako lineární kombinací *K* vlastních vektorů. Každý normalizovaný obličej je reprezentován jako vektor
$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \vdots \\ w_K^i \end{bmatrix}, i = 1,2, \dots, M .$$

Je-li eigenface použit pro detekci obličejů, pak lze vstupní obraz označit jako Γ. Samotná detekce pak sestává z několika kroků:

1. Od vstupního obrazu nejdříve odečteme průměrný obraz
$$\phi = \Gamma - \psi .$$
2. Vstupní obraz je třeba projektovat do prostoru obličejů
$$\hat{\phi} = \sum_{i=1}^{K} w_i u_i ,$$
$w_i = u_i^T \phi$, kde u_i^t je vlastní vektor výsledných eigenface
3. Dále je nutné vypočítat vzdálenost
$$e_d = \left\| \phi - \hat{\phi} \right\| .$$
4. Pokud $e_d < T_d$, pak Γ je obličej, kdy e_d je vzdálenost od prostoru obličeje a T_d je práh

Výhodou metody Eigenface je především možnost využití nejenom při detekci obličeje v obraze, ale také při rozpoznávání. Je-li k dispozici množina tváří, není problém zjišťovat vzdálenosti mezi vstupním obličejem a jednotlivými obličeji v databázi. Vybrán je obraz s tváří, jejíž vzdálenost k vstupní je nejmenší.

3. Viola a Jones algoritmus

3.1 Strojové učení

Strojové učení se zabývá algoritmy a postupy, které umožní umělému systému učit se, což znamená, že stroj bude mít schopnost automaticky zlepšovat svou výkonnost v závislosti na jeho vzrůstajících schopnostech a zkušenostech. Ve vztahu k detekci objektů, je možno stroji zlepšit jeho detekční schopnosti tím, že je mu předkládána sada obrazů s požadovaným objektem a dále obrazy, na nichž se objekt nevyskytuje. Výsledkem je, že stroj bude schopen na základě toho, co se naučil rozhodnout, zda v určitém obraze je daný objekt. A to i přesto, že to nebude obraz z trénovací sady obrázků, neboť detekce bude záviset na vybraných charakteristických vlastnosti objektu.

- **Učení s učitelem** je metoda, která má k dispozici množinu objektů, kterou lze zapsat do matice *D*

$$D = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & & x_{2m} \\ \vdots & & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}.$$

  Řádky matice D představují sledované objekty a sloupce jsou atributy příslušející jednotlivým objektům. Přidáním cílového atributu *y* do matice je možné řešit klasifikační úlohu

$$D = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} & y_1 \\ x_{21} & x_{22} & & x_{2m} & y_2 \\ \vdots & & \cdots & \vdots & \\ x_{n1} & x_{n2} & \cdots & x_{nm} & y_n \end{bmatrix}.$$

  Účelem je nalézt klasifikátory, které umožní použité vstupní množině přiřadit nějakou cílovou hodnotu $\hat{y}$

$$\hat{y} = f(x)\,.$$

  Hodnota $\hat{y}$ se bude samozřejmě lišit od skutečné hodnoty *y*, jinými slovy bude přítomná určitá chyba. Vhodným nalezením klasifikátorů je však možné tuto chybu minimalizovat.

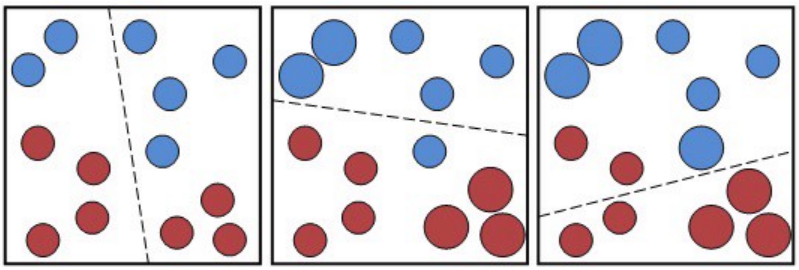
- **Učení bez učitele** se liší od předchozí metody absencí množiny dat k učení. Všechna vstupní data jsou považována za náhodná. Přesto je možné data členit získáním množiny vybraných charakteristických vlastností. K jejich získání je zapotřebí určitý počet testovaných objektů. Čím více se objekty liší, tím méně je objektů zapotřebí oproti situaci, kdy jsou vlastnosti méně výrazné.

3.2 AdaBoost

Algoritmus AdaBoost (Adaptive Boosting), popsaný Yoavem Freundem a Robertem Schapirem, slouží k zlepšení klasifikační přesnosti. Principem algoritmu je vytvoření konečné množiny slabých klasifikátorů, kterými lze pak daný objekt identifikovat, na základě vstupní množiny, které obsahuje pozitivní i negativní vzory pro trénování. Jedná se tedy o metodu učení s učitelem.

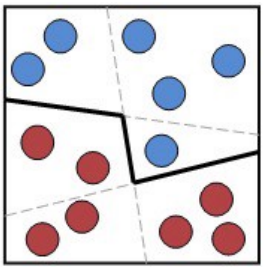
Jediným kladeným požadavkem na nově vzniklé slabé klasifikátory je, aby chyba byla menší než 0,5. Pokud by tato podmínka nebyla splněna, nemusí být zcela jistě zaručeno, že algoritmus v dalším postupu bude konvergovat.

Z určitého množství slabých klasifikátorů je nakonec vytvořen silný klasifikátor, jehož chyba je menší než chyba jednotlivých slabých klasifikátorů, čili došlo k zesílení klasifikace. Princip určování slabých klasifikátorů je na obrázku 3-1, kdy jsou postupně vybírány slabé klasifikátory. A vždy v následném kroku jsou špatně zařazené prvky dle použitého algoritmu převáženy.



Obrázek 3-1 Vytvoření slabých klasifikátorů

Jakmile jsou zjištěny slabé klasifikátory, je možné přistoupit k dalšímu kroku. V tom dojde k tomu, že kombinací slabých klasifikátorů, jež byly vytvořeny v předešlých krocích, se vytvoří silný klasifikátor.



Obrázek 3-2 Vytvoření silného klasifikátoru

Princip fungování popisovaného algoritmu AdaBoost je možné popsat pomocí následujícího pseudokódu.

Algoritmus 3-1 Adaboost

Vstup: $(x_1, y_1), x \in X, y \in Y, Y = \{+1, -1\}$
Inicializace $D = \frac{1}{m}$ pro všechny prvky
pro *t = 1 ... T*

1. Určení slabých klasifikátorů. Jeho nalezení tak, aby měl nejmenší chybu ε_j pro distribuci D_t.
2. Nalezení klasifikátoru s nejmenší chybou ε_t na distribuci D_t.

$$h_t: X \to [-1, +1]$$

3. Výpočet váhy klasifikátoru

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$$

4. Aktualizace vah vzorků ve vstupní množině

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

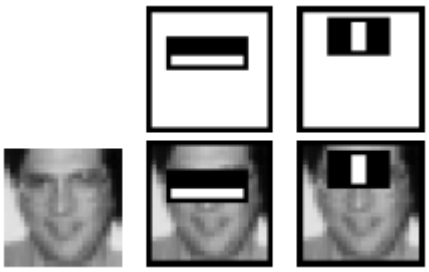
Kde Z_t je normalizační factor zvolený tak, aby D_{t+1} zůstala pravděpodobnostním rozložením, tj. $\sum_{i=0}^{m} D_t(i) = 1$.

Výsledný klasifikátor vznikne lineární kombinací nalezených slabých klasifikátorů. Hodnota funkce $H(x)$ po T krocích, během kterých bylo vytvořeno T slabých klasifikátorů, je dána součtem funkcí klasifikátorů násobených váhou

$$H(x) = \text{sign}\left(\sum_{t=0}^{T} \alpha_t h_t(x)\right).$$

3.3 Haarovy příznaky

Haarovy příznaky (Haarovy vlnky) jsou prvky, pomoci kterých se vypočítávají odezvy klasifikátorů, jak při trénování, tak i při detekci. Aplikování příznaků si lze představit tak, že vybraný prvek z dané množiny použitých Haarových příznaků pokryje oblast libovolného vstupního obrazu, jak je naznačeno na následujícím obrázku.



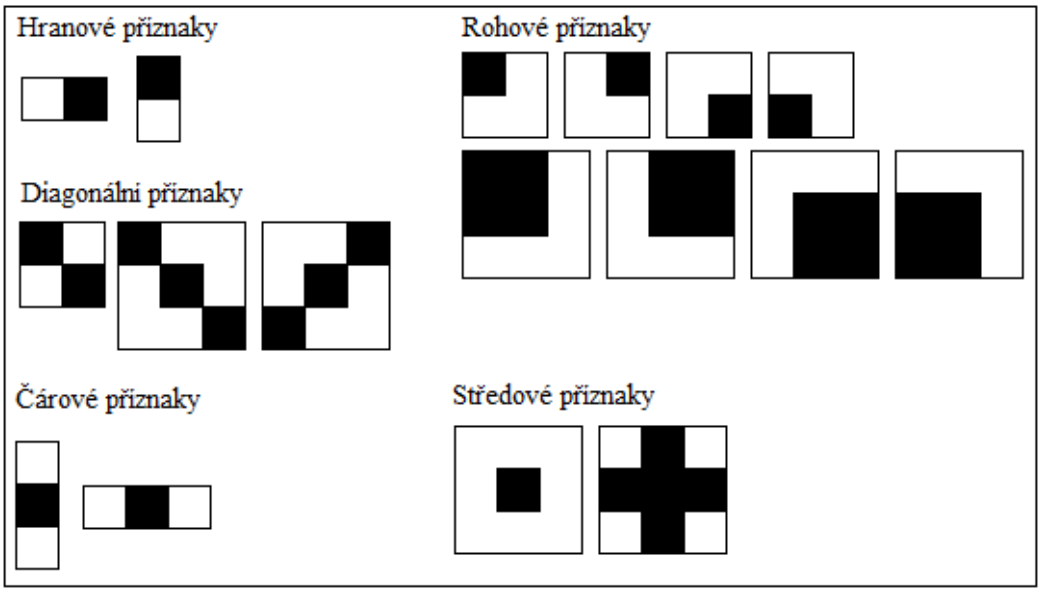
Obrázek 3-3 Aplikace Haarových příznaků [1]

Funkce neboli odezva příznaku je pak dána rozdílem součtů pixelů v jednotlivých oblastech

$$f(x) = \sum_{w \in W} x(w) - \sum_{b \in B} x(b).$$

Pro všechny pixely, které jsou pokryty vybraným příznakem, platí, že buď náleží do množiny W, nebo B. Jinými slovy se mohou vyskytovat pouze pod černou nebo bílou ploškou příznaku. Je-li navíc použitý obraz před samotným zjišťováním funkcí příznaků převeden do integrálního obrazu, je výpočet prováděn v konstantním čase, přičemž nebude záležet na velikosti původního obrazu ani na velikosti či pozici příznaku.

Některé vybrané Haarovy příznaky, které je při detekci objektů možno použít, jsou uvedeny na obrázku 3-4.



Obrázek 3-4 Vybrané Haarovy příznaky

Různých kombinací vybraných příznaků bude záviset na velikosti obrazu. Například pro trénink detektoru na obraze rozměru 24 x 24 pixelů s využitím pouze čtyř základních Haarových příznaků, kdy celkový počet všech kombinací příznaků dosahuje hodnoty 45396 [1]. Je tedy zřejmé, že tento výsledek je závislý na rozměru vstupní množiny obrazů i počtu příznaků, čímž v konečném důsledku bude růst i čas pro výpočet. Nicméně během procesu trénování je z tohoto velkého množství vybírána pouze malá množina charakteristických příznaků, která dokáže ve vstupním obraze při detekování hledaný objekt nejlépe rozpoznat.

3.4 Integrální obraz

Ze způsobu výpočtu odezvy Haarových příznaků je nutné znát součet pixelů pro libovolné oblasti vstupního obrazu. Intuitivně lze tohoto dosáhnout nejjednodušeji tak, že v závislosti na pozici a velikosti aktuálně testovaného příznaku, bude algoritmus celou pokrytou oblast procházet a sčítat hodnoty pixelů, které se zde nacházejí. V případě velkých oblastí je tento přístup nepříliš vhodný, neboť procházení a sčítání velkého množství pixelů bude zbytečně prodlužovat celý výpočet. Navíc je zjišťování odezvy Haarových příznaků velmi často používaná operace, jak při samotném trénování, tak i při detekci, a tudíž je vhodné tuto operaci optimalizovat. Řešení se nabízí v podobě vytvoření integrálního obrazu z originálního vstupního obrazu.

Integrální obraz vzniklý z původního obrazu o rozměrech *m x n* je *(m + 1) x (n + 1)* rozměrné pole, kde hodnota každého prvku o indexu I je dána jako součet pixelů, které jsou v původním obrazu v intervalu *(0...I – 1) x (0...I – 1)*, tedy

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y') \,.$$

Vytvoření integrálního obrazu může probíhat v jednom průchodu obrazu použitím následujících vzorců:

$$s(x,y) = s(x,y-1) + i(x,y) \,,$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \,,$$

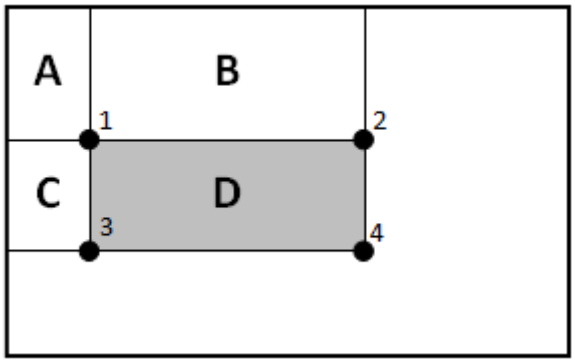
kde $s(x,y)$ je součet v sloupci, $s(x,-1) = 0$ a $ii(-1,y) = 0$.

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

0	0	0	0	0
0	1	2	3	4
0	2	4	6	8
0	3	6	9	12
0	4	8	12	16

Obrázek 3-5 Na prvním obrázku původní data a na druhém vzniklý integrální obrazu

Samotná odezva Haarova příznaku lze již jednoduše vyčíslit, jak ukazuje obrázek 3-6.



Obrázek 3-6 Součet pixelů v integrálním obrazu

Hodnota v bodě 1 je součet pixelů v oblasti *A*, hodnota v bodě 2 je dán jako *A* + *B*. Pro bod 3 lze hodnotu určit jako součet oblastí *A* + *C* a v bodě 4 je hodnota rovna součtu všech čtyř oblastí *A* + *B* + *C* + *D*. Ze znalosti hodnot v rohových bodech zkoumané oblasti *D* je již možné určit součet pixelů v této oblasti: 4 + 1 - (2 + 3).

Příklad

Je dán vstupní obraz z obrázku 3-5, kdy chceme znát součet pixelů v celé této oblasti použitím integrálního obrazu, jež je znázorněn na témže obrázku, bude použito výpočtu, jež je uveden výše. Pomocí hodnot v okrajových bodech, zjistíme, že 16 + 0 – (0 + 0) = 16.

■

Současně s integrálním obrazem se používá také integrální obraz druhých mocnin. Důvodem je určení standardní odchylky, která je pak využívána při výpočtu odezvy příznaku.

Integrální obraz druhých mocnin je možné vypočítat jako

$$ii(x,y) = \sum_{x' \le x, y' \le y} i^2(x',y') \,.$$

Nyní je možné určit na základě hodnot integrálního obrazu a integrálního obrazu druhých mocnin standardní odchylky obrazu

$$\sigma^2 = \frac{1}{N}\big(A + D - (B + C)\big) - \left[\frac{1}{N}\big(E + H - (F + G)\big)\right]^2 .$$

Kde *A, B, C, D* jsou hodnoty pixelů v rozích integrálního obrazu, dále *E, F, G, H* jsou hodnoty pixelů v rozích integrálního obrazu druhých mocnin a *N* je celkový počet pixelů v obrazu.

3.5 Viola a Jones algoritmus

Algoritmus Paula Violy a Michaela Jonese pro trénování a detekci obličejů byl poprvé použit roku 2001. Pro proces trénování je použita upravená verze algoritmu AdaBoost, který se oproti původnímu liší v inicializaci vah a také jejich aktualizaci. Algoritmus dále předpokládá použití Haarových vlnek jako klasifikačních příznaků a vytvoření integrálního obrazu z obrazu původního při trénování i detekci. Princip AdaBoost algoritmu, jenž byl pozměněn dle Violy a Jones popisuje následující pseudokód.

Algoritmus 3-2 Adaboost dle Viola a Jones [1]

Vstup: $(x_1, y_1), x \in X, y \in Y, Y = \{+1, -1\}$
Inicializace $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ pro $y_i = 0,1$
Pro *t = 1 ... T*

1. Normalizace vah, aby *w* bylo pravděpodobnostním rozložením.

$$w_{t,i} = \frac{w_{t,i}}{\sum_j w_{t,j}}$$

2. Určení slabých klasifikátorů. Nalezení optimálních parametrů, taka by měl co nejmenší chybu ε_j na aktuální distribuci *w*.

$$\varepsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$$

3. Výběr slabého klasifikátoru h_j s nejnižší chybou.

4. Aktualizace vah

$$w_{t+1,i} = w_{t,i}\beta_t^{1-\varepsilon_j}$$

Kde $\varepsilon_i = 0$ pokud vzorek *i* je klasifikován správně a $\varepsilon_i = 1$ pokud je klasifikován špatně. Platí, že $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$.

Hodnota silného klasifikátoru je dána

$$H(x) = \begin{cases} 1 \text{ pro } \sum_T \alpha_t h_t(x) > \frac{1}{2}\sum_T \alpha_t \\ 0 \text{ jinak} \end{cases},$$

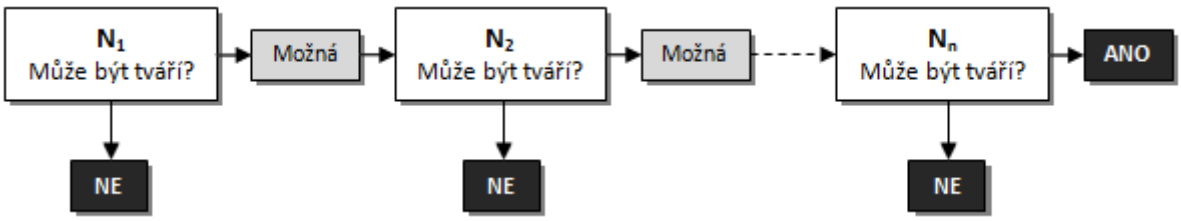
$$\alpha_t = \frac{1}{\beta_t}.$$

3.6 Viola a Jones algoritmus – kaskáda

Kaskádový algoritmus Violy a Jones přinesl vylepšení především týkající se rychlosti při detekování. Snížení doby výpočtu spočívá v předpokladu, že zpracovávaný obraz obsahuje mnohem větší množství oblastí, ve kterých se obličej nevyskytuje a není tak nutné jej procházet všemi klasifikátory.

Stupně kaskády jsou řazeny za sebou a každý z nich představuje jeden silný klasifikátor složený z určitého množství slabých klasifikátorů. Pokud dojde k pozitivní odezvě okna v určitém stupni kaskády, může se zde nacházet obličej a přejde se na další stupeň. Dané detekční okno je označeno, že obsahuje obličej pouze v případě, že dojde k pozitivní odezvě i posledního stupně celé kaskády.

Všechny stupně kaskády se tedy projdou pouze v případě, že aktuálně testované okno obsahuje obličej. Naopak v případě, že se v detekčním okně obličej nevyskytuje, dojde v některém ze stupňů kaskády k negativní odezvě, další stupně se již nezpracovávají a klasifikace pro dané okno se ukončí.



Obrázek 3-7 Viola a Jones algoritmus - kaskáda

Pro trénování jednotlivých silných klasifikátorů se používá předchozí algoritmus popsaný pomocí algoritmu 3-2.

Algoritmus 3-3 Trénování kaskády

1. **Vstup**:
   D … minimální povolený detekční poměr stupně kaskády
   F … maximální dovolený false positives rate na stupeň kaskády
   FPR … maximální dovolený false positives rate celého kaskádního klasifikátoru

P … pozitivní vzory
N … negativní vzory

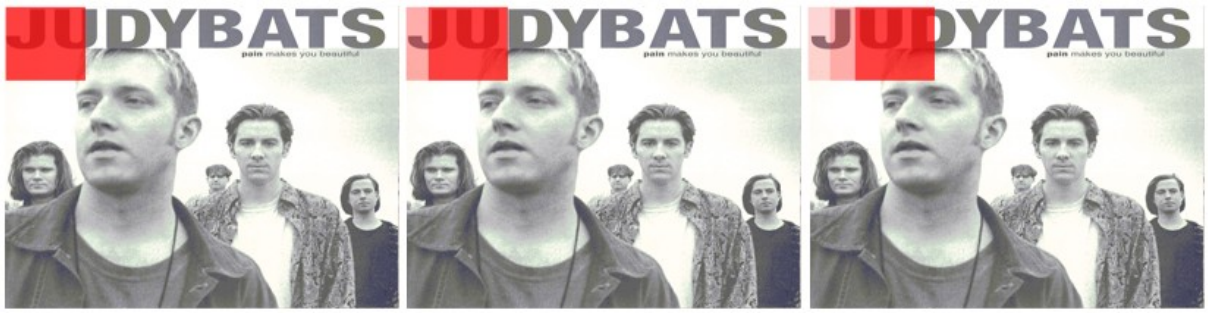
2. **Inicializace**
   $fpr_0 \leftarrow 1$
   $dr_0 \leftarrow 1$
   $i \leftarrow 0$

3. **Dokud** $F_i > F_{target}$
   a. $i \leftarrow i + 1$
   b. $n_i \leftarrow 0$; $F_i \leftarrow F_{i-1}$
   c. **Dokud** $F_i > f \times F_{i-1}$
      i. $n_i \leftarrow n_i + 1$
      ii. Natrénuj n_i slabých klasifikátorů (viz. Algoritmus 3-2)
      iii. Vyhodnoť aktuální kaskádou testovací množinu a zjisti fpr_i a dr_i
      iv. Sniž práh i-tého stupně kaskády dokud není dosáhnuto $dr_i \leftarrow D \times dr_{i-1}$
   d. $N \leftarrow \theta$
   e. **Jestliže** $F_i > F_{target}$ vyhodnoť současnou kaskádu na databázi negativních vzorů a všechny FP ulož do množiny N

3.7 Detekce obličejů

Jakmile je připraven soubor, jenž obsahuje naučené klasifikátory, je možné již detekovat obličeje v libovolném vstupním obraze. Základem celého procesu detekce je tzv. detekční okno, které prochází celý vstupní obraz. V každém detekčním okně se zjišťuje vyhodnocováním natrénovaných klasifikátorů, zda se v něm nenachází obličej.

Detekční okno se postupně posouvá celým obrazem a zjišťuje se, zda se v něm nachází obličej. Začátek celého procesu je znázorněn na obrázku 3-8. Jakmile dorazí do konce, dojde ke zvětšení detekčního okna vynásobením určitým koeficientem a opět se prochází celý vstupní obraz.



Obrázek 3-8 Procházení detekčních oken

Na vyhodnocení detekčního okna závisí, zda je dané okno označeno jako obličej či nikoliv. Pro každé okno se tak nejprve musí také vypočíst standardní odchylka, která je pak použita při výpočtu odezvy Haarova příznaku. Jelikož byly už dříve vytvořeny příslušné integrální obrazy, je tento výpočet proveden opět v konstantním čase, aniž by záleželo na velikosti okna.

Odezvu příznaku pro aktuálně zpracovávané detekční okno je možné vypočítat jako podíl funkce (odezvy) příznaku a součinu velikosti příznaku se standardní odchylkou

$$h(x) = \frac{f(x)}{size \cdot stddev}.$$

Nyní je možné spočítat odezvu slabého klasifikátoru, která je závislá právě na celkové odezvě příznaku. Tato získaná hodnota je porovnávána s vlastností threshold slabého klasifikátoru. Je-li odezva Haarova příznaku $h(x) \geq threshold$, pak je odezva slabého klasifikátoru vypočtena jako

$$w(x) = parity \cdot alpha\,,$$

ale naopak pokud je odezva příznaku $h(x) < threshold$, pak je odezva slabého klasifikátoru dána:

$$w(x) = -parity \cdot alpha\,.$$

Tento výpočet je postupně proveden pro *N* slabých klasifikátorů a celková odezva je dána součtem všech získaných hodnot

$$W(x) = \sum_{i=1}^{N} h_i(x)\,.$$

Délka výpočtu je závislá nejen na množství natrénovaných klasifikátorů, ale především na velikosti detekovaného obrazu. Je patrné, že čím bude vstupní obraz větší, tím více bude třeba vytvořit detekčních oken a výpočet se tak bude prodlužovat. V následující tabulce je uvedena závislost počtu vytvořených detekčních oken na koeficientu velikosti 1,2 pro obraz velikosti 144 x 192 pixelů:

Koeficient velikosti	**Počet detekčních oken**
1,2	20449
1,44	4897
1,72	4480
2,07	4081
2,48	1617
2,98	748
3,58	589
4,29	286
5,16	128
6,19	44
Celkem	**37319**

Tabulka 3-1 Závislost počtu detekčních oken na koeficientu velikosti

Koeficient zvětšení udává, o kolik procent se detekční okno zvětší oproti aktuálnímu stavu. Krok zvětšení ovlivní počet detekčních oken a může ovlivnit i detekční schopnost detektoru, protože v případě velkého rozdílu mezi velikostmi, může dojít k přeskočení oblastí, kde se vyskytuje obličej. Maximální hodnota koeficientu velikosti je ovlivněna rozměry vstupního obrazu, kdy strana detekčního okna nesmí přeskočit menší z dvou rozměrů obrazu.

4. Implementace učitele

V prvé řadě je nutno před samotným detekováním natrénovat množinu klasifikátorů, která bude rozpoznávat požadované objekty ve vstupních obrazech. Jako algoritmus pro trénování příznaků byl použit postup, jenž byl podrobněji popsán v kapitole 3. Pomocí implementovaného programu, lze trénovat více silných klasifikátorů zapojených do kaskády.

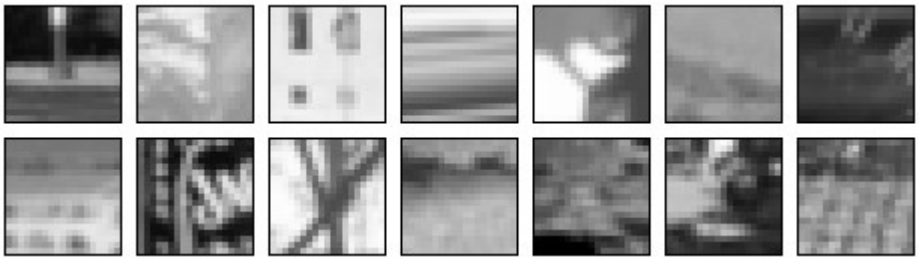
4.1 Zdrojová databáze vzorů

Vytvořená aplikace pro proces trénování používá volně dostupnou databázi MIT CBCL Face Database [12], která obsahuje pro trénování 2429 obrázků obličejů a 4548 obrázků, které obličej neobsahují. Jednotlivé obrázky v odstínech šedi mají rozměry 19 x 19 a jsou ve formátu PGM

Databáze pro trénování obsahuje pozitivní vzory:

Obrázek 4-1 MIT CBCL Face Database - obrazy obsahující obličeje [12]

A také negativní vzory:



Obrázek 4-2 MIT CBCL Face Database - obrazy neobsahující obličeje [12]

4.1.1 Formát PGM

Formát PGM (Portable GrayMap) je určený pro uchovávání obrazových dat ve stupních šedi. Použita databáze využívá binární variantu, která má vždy následující schéma.

```
P5
Šířka
Výška
Maximální hodnota pixelu
Binární rastrová data
```

Tabulka 4-1 Struktura binárního PGM

Vzhledem ke své jednoduchosti formát PGM umožňuje velmi jednoduché načítání potřebných obrazových dat.

4.2 Postup

Vstupem aplikace pro učení klasifikátorů jsou cesty k adresářům obsahující pozitivní vzory a negativní vzory. Další dva vstupy jsou číselné hodnoty, které udávají, kolik obrázků z dané množiny se mají použít pro trénování. Adresáře s obrázky musí obsahovat obrázky ve formátu PGM.

Všechny vstupní obrazy určené k učení by měly mít stejnou velikost. Důvodem je, že ještě před začátkem celého procesu se musí učit všechny možné kombinace Haarových příznaků, které lze vytvořit na obrázku o dané velikosti.

Výpočet počtu kombinací jednoho klasifikátoru je závislý na velikosti obrazu z vstupní množiny a příznaku *i*

$$ratioW = \frac{imageW}{featureW_i},$$

$$ratioH = \frac{imageH}{featureH_i},$$

$$x_i = \left[ratioW \cdot \left(imageW + 1 - \frac{featureW_i \cdot (ratioW + 1)}{2}\right)\right],$$

$$y_i = \left[ratioH \cdot \left(imageH + 1 - \frac{featureH_i \cdot (ratioH + 1)}{2}\right)\right].$$

Celkový počet je pak dán

$$feature_i = x_i \cdot y_i .$$

Příklad

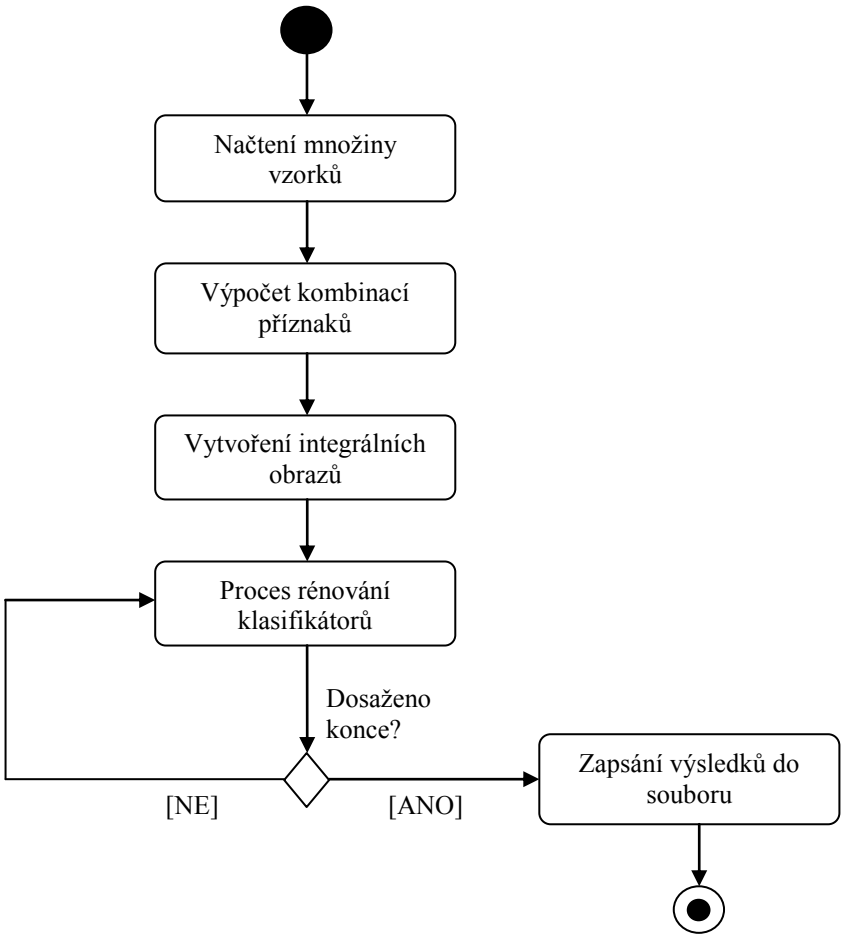
Celkový počet příznaků pro obraz velikosti 19 x 19 a pro první z Haarových příznaků z obrázku 3-4, jehož rozměry jsou 2 x 1, lze zjistit dosazením do uvedených vzorců. Tak zjistíme, že pro daný obraz je možné určit 17100 kombinací daného příznaku.

■

Lze vidět, že i na tak malý obraz připadnou ve výsledků velké množství Haarových příznaků. Tento počet navíc bude růst, pokud se zvětší obraz. Zvětší-li se šířka a výška obrazu o jeden pixel, pak počet kombinací pro vybraný Haarův příznak z předchozího příkladu, vzroste na 21000. Také se samozřejmě celkový počet kombinací všech příznaků zvětší, přidáme-li do množiny klasifikátorů další Haarův příznak.

Dále je při načítání každého vzorového obrazu vypočten integrální obraz a integrální obraz druhých mocnin spolu s určením standardní odchylky. Všechny informace jsou již připraveny jako vstup do učícího algoritmu. Jakmile jsou splněny podmínky pro ukončení procesu učení, výsledné klasifikátory a další potřebné hodnoty se uloží do souboru pro potřeby použití v detektoru.

Posloupnost jednotlivých kroků učícího procesu jsou ilustrovány na obrázku 4-3. Akce pojmenována jako trénování silných a slabých klasifikátorů v sobě zapouzdřuje některý z algoritmů pro trénování, který vytváří buď monolitický detektor, nebo kaskádový detektor.



Obrázek 4-3 Kroky při trénování

Celý proces trénování je časově velmi náročný. Dobu výpočtu vybírajícího jeden slabý klasifikátor ovlivňuje především velikost množiny obrázků a jejich velikost. Čím větší bude obrázek, tím více bude různých kombinací Haarových příznaků, které je nutné otestovat. Je-li však vyžadována vysoká úspěšnost detektoru, pak je nutné, aby se trénování provedlo na co možná největší množině. Například Eyedea Recognition Ltd. uvádí, že jejich detektor obličejů je natrénován na množině 500000 pozitivních a $4 \cdot 10^9$ negativních příkladů [11].

Jelikož se ukázalo pro použitou databázi, že trénování je zdlouhavé, byla z tohoto důvodu implementována více-vláknová modifikace oproti původně zamýšlené jedno-vláknové aplikace, která tak snížila celkovou dobu trénování.

Při provádění paralelního trénování klasifikátorů však nelze trénovat více klasifikátorů najednou každým vláknem, jelikož po výběru klasifikátoru s nejmenší chybou, dochází k převážení celé množiny. Kdybychom trénovali například čtyři klasifikátory najednou, pak bychom dostali ve výsledku čtyři stejné slabé klasifikátory, protože všechny vlákna pracují na stejné množině vstupních dat se stejnou váhou. Z tohoto důvodu byl při implementaci zvolen postup, kdy se vlákna podílejí na výběru jednoho klasifikátoru. Nicméně je nutné zajistit, aby nedošlo k převážení množiny dříve, nežli jsou ukončena všechna vlákna, která vybírají nejvhodnější příznak. Teprve po synchronizaci vláken je možné množinu převážit a normalizovat. Tento přístup je aplikován jak u vytváření monolitického klasifikátoru, tak u kaskádového detektoru.

Při implementaci trénování kaskády se objevil problém s používáním negativní množiny obrázků. Jak je popsáno v popisu algoritmu 3-3 pro každý další stupeň kaskády se používají prvky z množiny obsahující negativní vzory, které byly v předcházejícím stupni označeny jako pozitivní vzory. Postupně se tak odebírají prvky z negativní množiny. Může dojít k situaci, kdy jsou všechny prvky odstraněny ještě dříve, než dojde k vytvoření požadované množiny klasifikátorů. Z tohoto

důvodu je žádoucí stanovit minimální hodnotu počtu negativních vzorků a v případě nutnosti je po vytvoření každého nového stupně kaskády doplňovat.

Aby se snížilo riziko nedostatku vzorků, bylo vytvořeno několik větších obrazů, do kterých je postupně uloženo větší množství prvků z negativní množiny. Obraz vytvořený z negativních vzorů je na obrázku 4-4:

Obrázek 4-4 Obraz negativních vzorků

Vždy před započetím vytváření nového stupně kaskády jsou posléze z těchto obrazů náhodně získávány obrazová data podle následujícího algoritmu:

Algoritmus 4-1 Vybírání nových negativních vzorků

1. **Vstup**:
   N … množina negativních vzorků určených pro trénování
   Negatives … množina obrazů s negativními vzory
   countNegatives … počet prvků v množině Negatives
   nwidth … šířka obrazu v množině Negatives
   nheight … výška obrazu v množině Negatives
   sample_width … šířka vzorku
   sample_height … výška vzorku
   actualNegatives … aktuální počet negativních vzorků
   minimumNegatives … minimální počet negativních vzorků

2. **Dokud** *actualNegatives* < *minimumNegatives*
   a. *indexN* ← *rand (0 ... countNegatives)*
   b. *startX* ← *rand (0 ... nwidth – sample_width)*
   c. *startY* ← *rand (0 ... nheight – sample_height)*
   d. Vytvoř vzorek o velikosti *sample_width* × *sample_height* vybraný z množiny obrazů *Negatives* na pozici *indexN*. Na základě hodnot *startX*, *startY* a rozměrů *sample_width*, *sample_height* proveď výřez obrazových dat. Dále urči integrální obraz, integrální obraz druhých mocnin a standardní odchylku výřezu.
   e. **Jestliže** se jedná o první stupeň kaskády

i. Vlož ihned do množiny N
ii. *actualNegatives* ← *actualNegatives + 1*

f. **Jinak**

i. otestuj nový vzorek na aktuálním kaskádovém detektoru.
ii. **Jestliže** vzorek je označen jako hledaný objekt
1. vlož vzorek do množiny *N*
2. *actualNegatives* ← *actualNegatives + 1*
iii. **Jinak**
1. Uvolni výřez z paměti a pokračuj znovu od začátku cyklu

Výstupem programu je soubor natrénovaných klasifikátorů ve formátu XML. Do tohoto souboru se ukládá velikost vzorových obrázků použitých při trénování, aby se z nich určila minimální velikost detekčního okna. Dále se ukládají jednotlivé atributy slabých klasifikátorů obsahující Haarovy příznaky, polaritu, chybu a práh. Jedná-li se o uložení kaskády, jsou v něm obsaženy tyto slabé klasifikátory a navíc je opatřen prahovou hodnotou. Právě tento XML soubor je pak používán detektorem při detekci daného objektu, kdy aplikuje jednotlivé klasifikátory obsažené právě v tomto souboru.

5. Implementace detektoru

Detekční program bude mít několik úkolů. Nejprve musí vstupní obraz převést do integrálních obrazů a určit z nich standardní odchylku. Následně připraví detekční okna a provede v nich detekci, která je teoreticky popsána v kapitole 3. Získané výsledky nakonec musí ve vhodné podobě zobrazit uživateli.

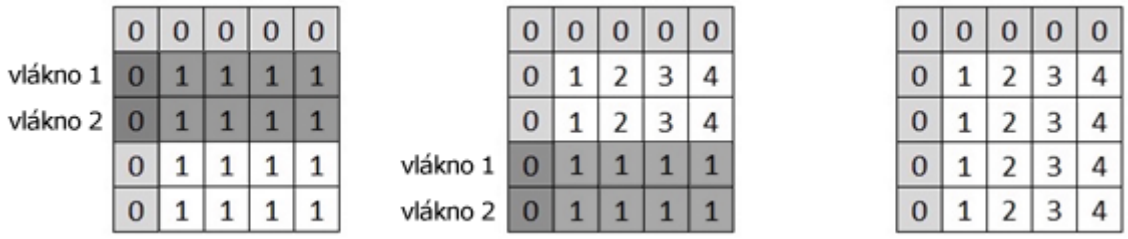
5.1 Paralelní výpočet integrálního obrazu

V případě, kdy jsou na vstupu velká vstupní data, je možné urychlit paralelním výpočtem integrální obraz. Toto se týká především detekce, jelikož právě u ní je možné předpokládat i větší vstupní data. Při trénování jsou použité obrazy mnohem menší, a proto zde paralelní výpočet není příliš potřebný.

Algoritmus 5-1 Výpočet integrálního obrazu

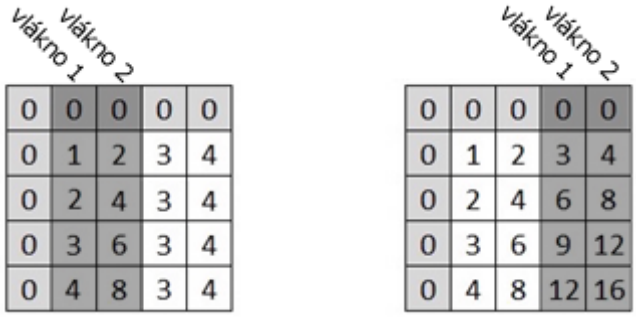
Vstup:
Pole *data* obrazových dat, rozměry obrazu *width* a *height*.
Pro *h*←*1 ... height*
Pro *w*←*1... width*
1. ii[w, h] ← ii[w, h-1] + ii[w-1, h] + data[w, h] – ii[w-1, h-1]

Z výše uvedeného pseudokódu je možné určit, jak může paralelní zpracování probíhat. Princip spočívá v rozdělení výpočtu na dvě části. Jelikož se výsledky sčítání v řadách navzájem neovlivňují, což platí i pro sloupce, je možné nejprve vypočítat sumy pixelů v řadách a z takto vzniklých nových dat dopočítat sumy ve sloupcích. Při výpočtu se pak bude zpracovávat souběžně více řad či sloupců.
Nejprve tedy dojde k paralelnímu zpracování všech řádků:



Obrázek 5-1 Paralelní výpočet integrálního obrazu – zpracování řádků

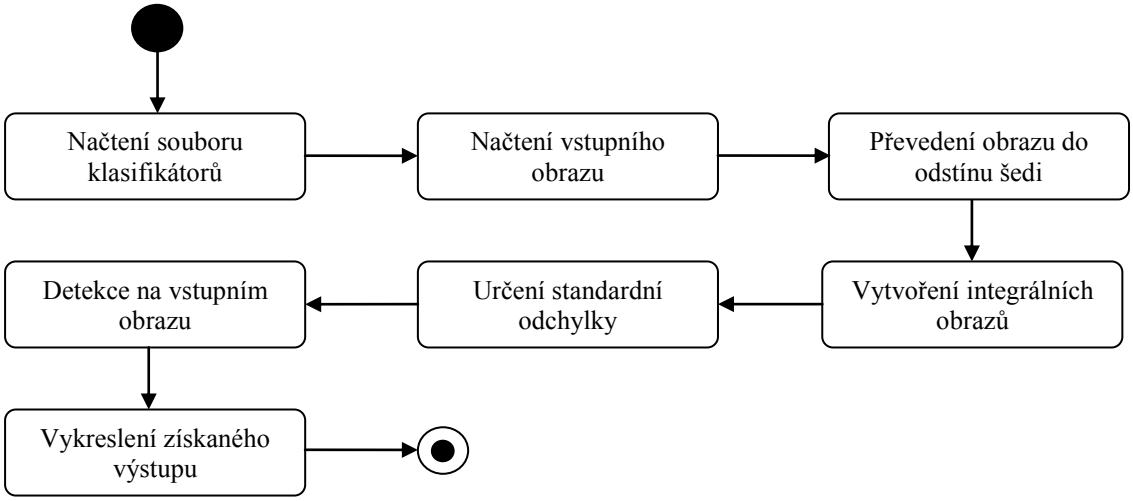
Výsledný integrální obraz je získán v okamžiku, kdy jsou zpracovány paralelně jednotlivé sloupce:



Obrázek 5-2 Paralelní výpočet integrálního obrazu - zpracování sloupců

5.2 Postup

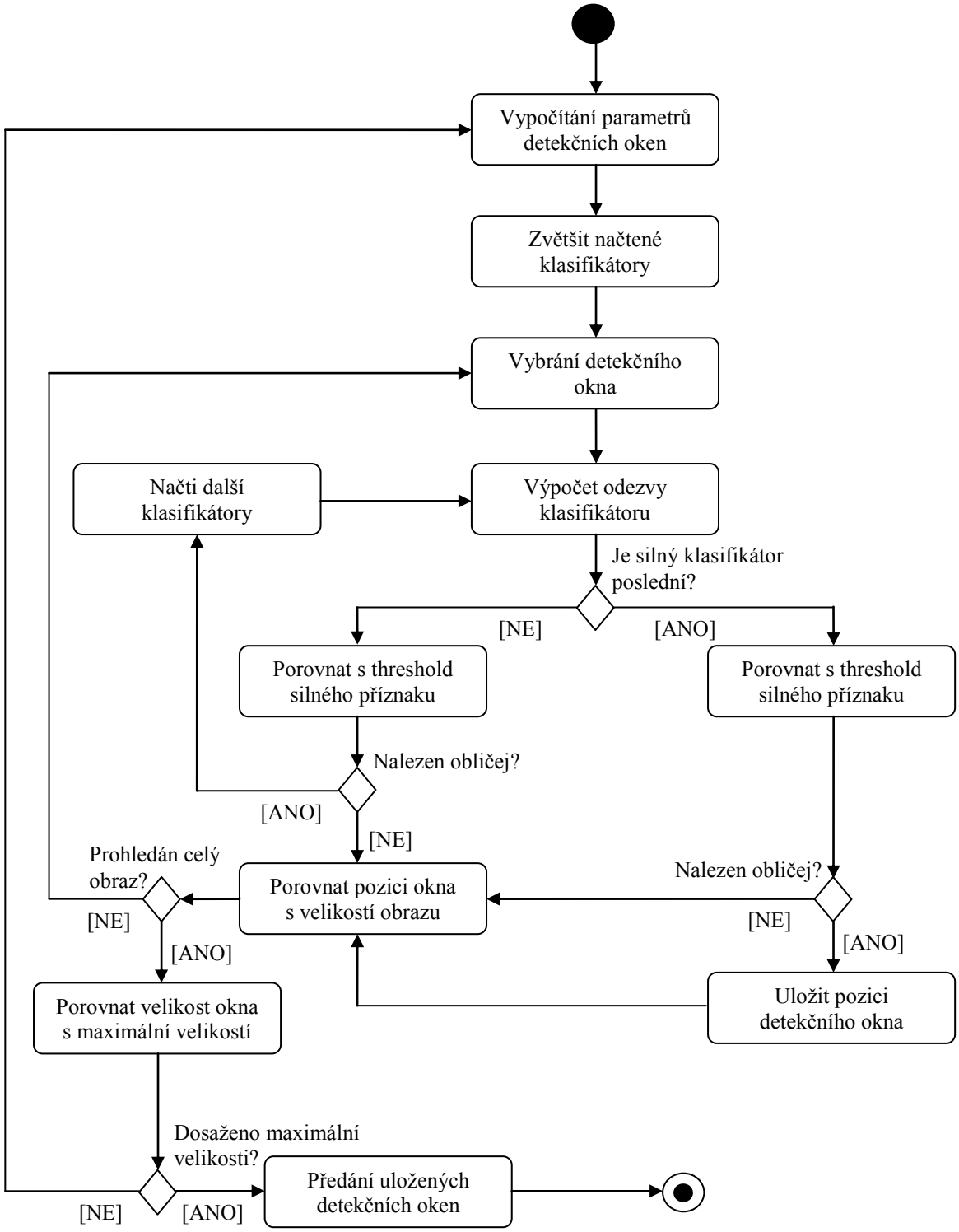
Samotná detekce sestává z několika kroků, kdy je nejprve načten soubor s klasifikátory a vstupní obraz, na kterém hledáme obličeje. Ten je poté převeden do odstínu šedi a dále se vytvoří integrální obraz a integrální obraz druhých mocnin pro výpočet standardní odchylky. Až poté je již možné provést samotnou detekci, přičemž jsou zpracovávány všechny části obrazu, tzv. detekční okna, jejichž minimální rozměr má rozměr obrazů, které byly použity při trénování klasifikátorů. Jednotlivé kroky jsou na obrázku 5-3.



Obrázek 5-3 Kroky při detekci

Detekce je časově náročná zejména z důvodu velkého množství detekčních oken (viz. Tabulka 3-1). Během procesu rozpoznávání určitého objektu je tedy zapotřebí projít všechna detekční okna v různých velikostech. Okno se postupně jakoby posouvá po obraze určitým krokem. Velikost tohoto kroku může předem definovaná. Není však vhodné volit velký krok, který by sice vedl k urychlení celého procesu detekce, ale mohlo by dojít k přeskočení částí obrazů, ve kterém by se mohli nacházet objekty, jež chceme rozpoznat. Navíc je nutné nastaveným velikostním koeficientem při každém zvětšování okna také zvětšit celou načtenou množinu klasifikátorů.

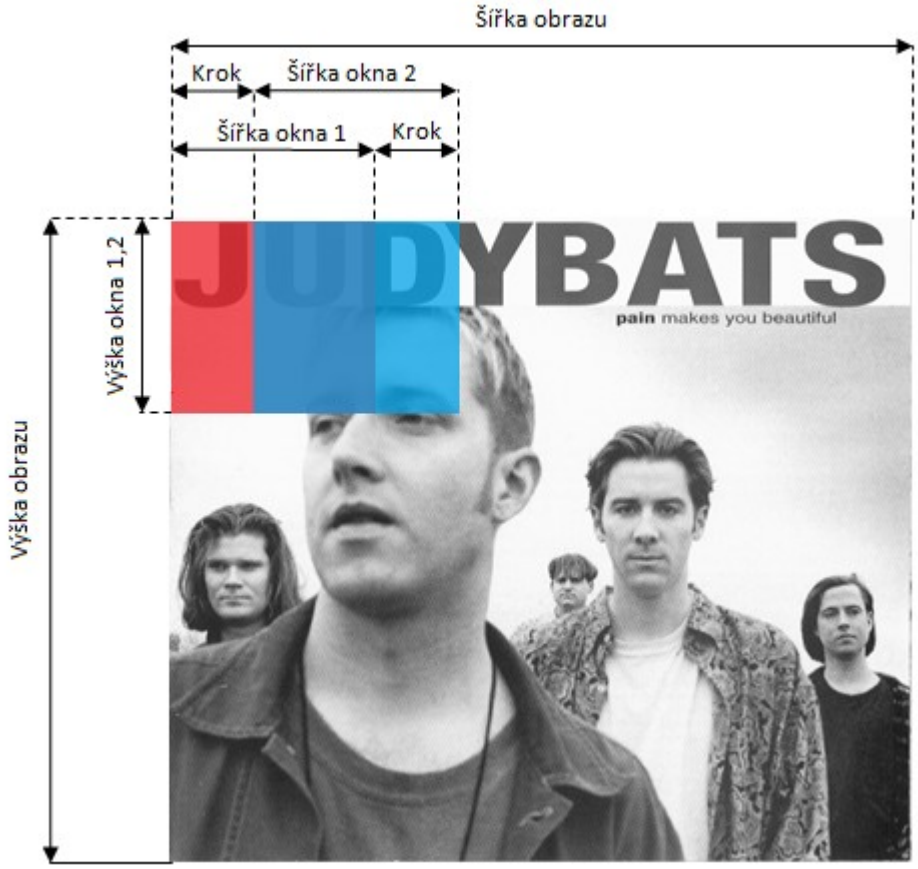
Teoreticky byl popsán postup detekce v kapitole 3, kdy je na každé okno aplikována určitá množina natrénovaných klasifikátorů. Detailněji, pomoci diagramu, je celý průběh detekce zachycen na obrázku 5-4.



Obrázek 5-4 Průběh detekování

Při více-vláknovém pojetí není vhodné přepočítávat pozice a velikosti detekčních oken. Především u GPU implementace se ukázalo, že je nevhodné vypočítat detekční okna předem na CPU, a následně je posílat před započetím detekce na GPU. Dalším aspektem proč nemít okna uložena v paměti, je právě jejich možné velké množství.

Z těchto důvodů dochází k výpočtu pozice aktuálního detekčního okna a velikosti přímo za běhu na základě indexu okna. Výpočet je poněkud složitější z důvodu volitelného kroku mezi detekčními okny. Avšak i tak se během testování ukázalo, že tento postup je rychlejší, především při detekování přes GPU, kdy odpadají náročné přesouvání bloků dat s detekčními okny do paměti grafické karty.



Obrázek 5-5 Průběh detekčního okna v obraze

Ze zadaných údajů jsou na základě indexu okna vypočítány rozměry a pozice levého horního rohu. Ze zjištěné šířky obrázku a šířky detekčního okna lze nejprve určit maximální pozici okna v jednom řádku, která odpovídá rozdílu šířky obrazu a detekčního okna. Důvodem je vyhnutí se překročení šířky obrazu a analogický výpočet se provede i pro výšku. Dále potřebujeme počet možných oken pro každý řádek a sloupec. Tento údaj získáme pro horizontální posun podílem maximální x-ové pozice a kroku a stejně tak podílem nejvyšší y-ové pozice a kroku pro vertikální posun.

Všechny údaje popisující sadu detekčních oken je nutné aktualizovat při každé změně velikosti detekčního okna. Zvětšením okna příslušnou hodnotou koeficientu se mění všechny informace, které jsme získávali.

Dojde-li pak k samotné detekci, každé detekční okno je opatřeno svým indexem, který je odvozen od indexu vlákna, které v daný moment zpracovává. Dle hodnoty indexu okna se nakonec určí x-ová a y-ová souřadnice levého horního bodu a nakonec se ze získaných hodnot a rozměrů okna mohou určit i souřadnice pravého spodního rohu.

Ze všech údajů je již možné určit indexy čtyř okrajových pixelů ve vstupním obrazu a tím i vlastně všechny pixely, které se v dané ohraničené oblasti nacházejí. Nic už nyní nebrání tomu, aby se mohlo pro dané okno provést detekce tváře.

Strukturou Rectangle je popsáno jak detekční okno při detekci a zároveň popisuje i čtvercové oblasti v Haarových příznacích:

```
struct Rectangle
{
   int x, y, w, h;
   int p1, p2, p3, p4;
};
```

Výpis 5-1 Struktura Haarova příznaku

Kromě informací o velikosti a pozici levého horního rohu, obsahuje navíc čtyři indexy, které určují čtyři rohové pozice příznaku v obraze. K tomuto slouží následující funkce, která na základě rozměrů okna a šířce obrazu spočítá příslušné indexy:

```
void UpdateIndexPointers(int imageW)
{
   p1 = (x - 1) + imageW * (y - 1);
   p2 = (x - 1 + w) + imageW * (y - 1);
   p3 = (x - 1) + imageW * (y - 1 + h);
   p4 = (x - 1 + w) + imageW * (y - 1 + h);
}
```

Výpis 5-2 Aktualizace indexů v obrazu

Struktura Rectangle je navíc používána i pro vytváření Haarových příznaků. Každý takový příznak je vytvořen ze dvou částí, představující černou a bílou plochu. Aby se rozlišily, je navíc přidána vlastnost weight.

```
struct HaarFeature
{
   Rectangle rect[2];
   float weight[2];
};
```

Výpis 5-3 Struktura Haarova příznaku

Před započetím procesu učení, jsou nejprve vytvářeny jednotlivé příznaky, které se budou na vstupní množině trénovat. Příklad vytvoření jednoho Haarova příznaku:

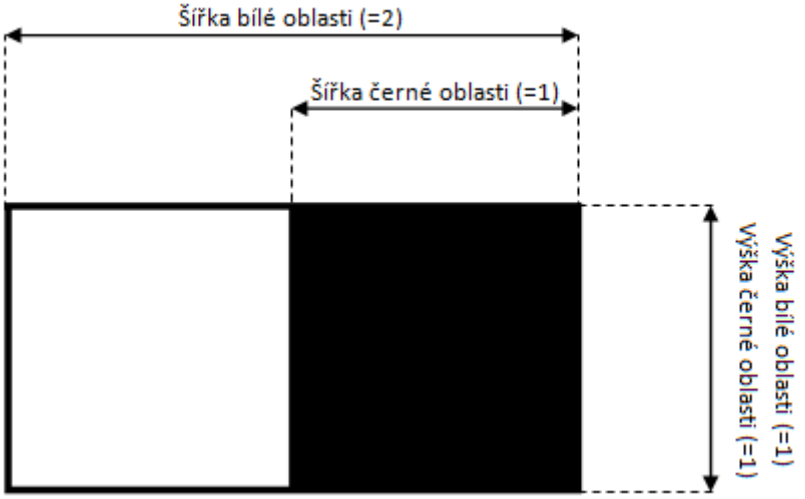
```
HaarFeature* feature = (HaarFeature*)malloc(sizeof(HaarFeature));

feature->rect[0].x = 0;
feature->rect[0].y = 0;
feature->rect[0].w = 2;
feature->rect[0].h = 1;
feature->weight[0] = -1.0f;

feature->rect[1].x = 1;
feature->rect[0].y = 0;
feature->rect[0].w = 1;
feature->rect[0].h = 1;
feature->weight[1] = 2.0;
```

Výpis 5-4 Vytvoření konkrétního Haarova příznaku

Výše uvedený kód vytváří následující Haarův příznak:



Obrázek 5-6 Vytvořený Haarův příznak dle uvedeného kódu

Při definování jsou z rozměrů příznaku automaticky spočteny indexy okrajových oblastí, pokryté daným příznakem. Používá se funkce popsána ve výpisu 5-2, kdy vstupním parametrem je šířka trénovacího obrazu.

Na odezvu Haarova příznaku je důležitá funkce z výpisu 5-5, která mimo jiné názorně ukazuje použití přímých indexů do obrazu, které byly dříve při definici příznaku přepočítány. Vstupem do této funkce je obrázek, testované detekční okno, velikostní koeficient a standardní odchylka obrazu.

```
float GetFeatureValue(Image* img, Rectangle r, float scale, float stdvar)
{
   float retValue = (img->invstdvar) *
     ( (img->ii[rect[0].p1] + img->ii[rec[0].p4]
       - img->ii[rec[0].p2] - img->ii[rec[0].p3]) * weight[0]
     + (img->ii[rect[1].p1] + img->ii[rec[1].p4]
       - img->ii[rec[1].p2] - img->ii[rec[1].p3]) * weight[1]
     );

   return retValue/(odvar * scale * scale);
}
```

Výpis 5-5 Odezva Haarova příznaku

Haarův příznak je hlavní složkou slabého klasifikátoru. Navíc však obsahuje další parametry, které jsou použity k detekci.

```
struct WeakClassifier
{
   HaarFeature feature;
   float threshold;
   float alpha;
   bool polarity;
};
```

Výpis 5-6 Struktura slabého klasifikátoru

Při získání odezvy slabého klasifikátoru se získává odezva Haarova příznaku zavoláním příslušné funkce. Získaná hodnota je porovnána s atributem threshold slabého klasifikátoru a nakonec je určena návratová hodnota.

```
float GetFeatureValue(Image* img, Rectangle r, float scale, float stdvar)
{
   if(feature.GetFeatureValue(img, r, scale, stdvar) < threshold)
   {
      return polarity ? alpha : -alpha;
   }
   else
   {
      return polarity ? -alpha : alpha;
   }
}
```

Výpis 5-7 Odezva slabého klasifikátoru

Silný klasifikátor obsahuje konečnou množinu slabých klasifikátorů a threshold. Jeho struktura je v následujícím výpisu:

```
struct StrongClassifier
{
   WeakClassifier* weaks;
   unsigned int nweak;
   float threshold;
};
```

Výpis 5-8 Struktura silného klasifikátoru

Pro získání odezvy silného klasifikátoru je nutné projít všechny slabé klasifikátory k získání jejich odezvy. Na základě porovnání získané hodnoty s thresholdem je vrácena příslušná hodnota.

```
float GetFeatureValue(Image* img, Rectangle r, float scale, float stdvar)
{
   float retValue = 0.0f;
   for(unsigned int i=0; i<nweak; i++)
   {
      retValue += weaks[i].GetFeatureValue(img, r, scale, stdvar);
   }

   if(retValue < threshold)
   {
      return -1.0f;
   }
   else
   {
      return 1.0f;
   }
}
```

Výpis 5-9 Odezva silného klasifikátoru

Posledním prvkem je samotný klasifikátor, který obsahuje threshold a silný klasifikátor. V případě kaskádového detektoru pak množinu silných klasifikátorů a jejich celkový počet, který je třeba znát pro výpočet odezev.

```
struct classifier
{
   StrongClassifier* strongs;
   unsigned int nstrong;
   float threshold;
};
```

Výpis 5-10 Struktura klasifikátoru

Klasifikátor získává odezvu silného klasifikátoru, která je porovnávána s thresholdem. Jakmile některý silný klasifikátor rozhodne, že se v daném detekčním okně nenachází hledaný objekt, je zaslána návratová hodnota -1.0 a další detekování již neprobíhá. V případě, že všechny silné klasifikátory rozhodnou, že se v daném okně nachází daný objekt, je vrácena 1.0.

```
float GetFeatureValue(Image* img, Rectangle r, float scale, float stdvar)
{
   for(unsigned int i=0; i<nstrong; i++)
      if(strong[i].GetFeatureValue(img, r, scale, stdvar) < threshold)
         return -1.0f;
   return 1.0f;
}
```

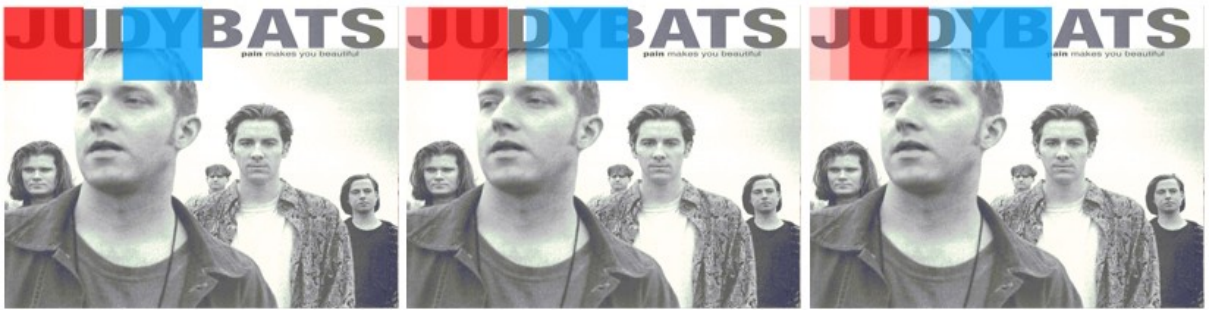
Výpis 5-11 Odezva klasifikátoru

5.3 Více-vláknový detektor

Jelikož se výsledné odezvy detekčních oken navzájem nijak neovlivňují, lze proces detekce jednoduše paralelizovat tak, že je ve stejný okamžik testováno více detekčních oken. Toto lze realizovat pomocí CPU i GPU.

V zásadě před započetím implementace byly zvažovány tři možnosti, jak paralelizovat detekční proces, přičemž je možné souběžně zpracovávat detekční okna, slabé klasifikátory, nebo celé sady detekčních oken dle velikostí. Jelikož hodnota odezvy klasifikátoru pro určité detekční okno nejsou na sobě závislé, je možné provádět paralelní výpočet odezvy více oken v jeden okamžik, což lze samozřejmě realizovat na CPU i na GPU. Problém nastává v okamžiku, kdy některé ze zpracovávaných detekčních oken aktuální velikosti dosáhne poslední pozice v obrazu. Pro další běh se tedy musí okno zvětšit spolu se všemi natrénovanými příznaky. Je nutné si ale uvědomit, že ne všechna vlákna musí nutně ukončit procházení dané velikosti ve stejný okamžik. Při změně velikosti příznaků, budou pak neukončená vlákna používat špatnou sadu, což vede ke špatným detekcím.

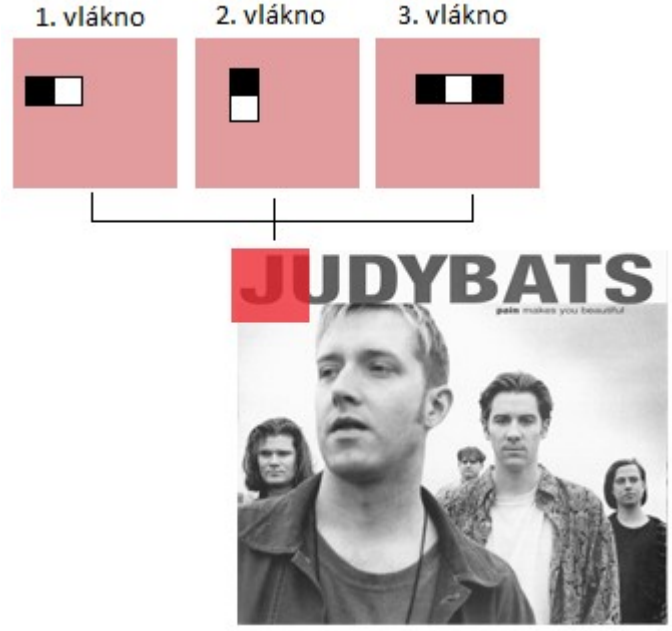
Jedním z řešení může být, že každé vlákno bude mít svou vlastní kopii klasifikátorů. Nicméně z hlediska paměťových nároků, toto nemusí být zcela nejlepší řešení. Proto bylo při implementaci zvoleno jiné řešení, které používá pouze jednu sadu klasifikátorů sdílenou všemi vlákny. Aby nedošlo k popsanému problému, jsou všechna vlákna synchronizována ve chvíli, kdy všechna zkontrolují své poslední detekční okna. Každé vlákno bude kontrolovat přibližně stejný počet oken, tudíž by nemělo docházet k dlouhému čekání na ostatní vlákna. Ve chvíli, kdy jsou vlákna synchronizována, dojde ke zvětšení detekčního okna a zároveň i příznaků a celý proces se opakuje znovu.



Obrázek 5-7 Paralelní zpracování detekčních oken dvěma vlákny

Druhou možností je paralelizovat jednotlivé klasifikátory. Detekční okna budou procházena sekvenčně, zatímco klasifikátory budou paralelně aplikovány na aktuálně vybrané okno. Výsledná odezva je pak dána součtem odezev. V případě, že je použit pouze jeden silný klasifikátor, celá sada slabých klasifikátorů může být paralelizována. Při kaskádové detekci jsou také zpracovávány souběžně jen slabé klasifikátory, ale pouze ty příslušející stejnému silnému klasifikátoru. Pokud pro daný silný klasifikátor bude výsledek pozitivní, nová sada slabých klasifikátorů bude přiřazena jednotlivým vláknům.

Problémem tohoto postupu je, že první stupně kaskády nemají obvykle velké množství slabých klasifikátorů. V případě, kdy bychom měli k dispozici větší množství vláken, než kolik je klasifikátorů, zůstaly by některá vlákna zcela nevyužité. Navíc právě většina oken, které neobsahují obličej, je vyřazena právě silnými klasifikátory, které se nacházejí na začátku celé kaskády. Naopak výhodou je, že odpadají problémy s různě velkými příznaky. V okamžiku, kdy by se zvětšila detekční okna, zvětšily by se i příznaky.



Obrázek 5-8 Paralelní zpracování klasifikátorů

Obrázek 5-8 ukazuje, jak by mohlo vypadat paralelní zpracování klasifikátorů. Dejme tomu, že bychom měli k dispozici tři vlákna. Do každého lze umístit jeden natrénovaný klasifikátor a ten v aktuálně vybraném detekčním okně otestovat. Výsledky odezvy z každého vlákna by se pak po aplikování každého klasifikátoru sčítali a na základě konečné hodnoty by se rozhodlo, zda se v detekčním okně nachází obličej.

Třetí možností, která byla před započetím implementace zvažována, byla metoda paralelního zpracování oken různých velikostí. Stejně jako u prvního zmíněného postupu je toto možné, protože neexistuje žádná závislost mezi různě velkými okny na stejné pozici. Pokud menší okno detekuje v určitém místě obličej, neznamená to, že zvětšené okno nutně detekuje ve stejném místě také obličej. Totéž platí naopak.

Každé vlákno by zpracovávalo sadu oken jedné velikosti, přičemž musí mít přístup i k vlastní sadě natrénovaných příznaků, které jsou zvětšené stejným velikostním koeficientem jako detekční okno. Problémem ovšem je, že se zvětšováním detekčních oken se zmenšuje jejich počet. V případě, kdy jsou velké rozdíly mezi počty detekčních oken v různých vláknech, znamená to, že některé vlákna se ukončí příliš brzy a budou nevyužitelně čekat na vlákna, která obsahují větší počty detekčních oken.

Obrázek 5-9 Paralelní zpracování oken dle velikosti

Pro příklad lze použít údaje z tabulky 3-1. Pro koeficient 1,2 je k dispozici 20449 oken, zatímco pro jednou zvětšené detekční okno s koeficientem 1,44 je to už pouze 4897. Výsledky z detekce prvního vlákna, které zpracovává 20449 oken, by bylo mnohem pomalejší než druhé vlákno zpracující pouze čtvrtinový počet oken. Pro toto řešení je nutné zajistit, aby vlákno, které už skončilo se všemi detekcemi, přešlo k detekcím další množiny detekčních oken. Stejně by však nebylo možné přesně synchronizovat vlákna a došlo by ve většině případů k tomu, že by se čekalo na vlákno, které ještě neprošlo všechna detekční okna. Z důvodu této komplikovanosti řešení nebyl volen tento způsob implementace.

6. Implementace na GPU

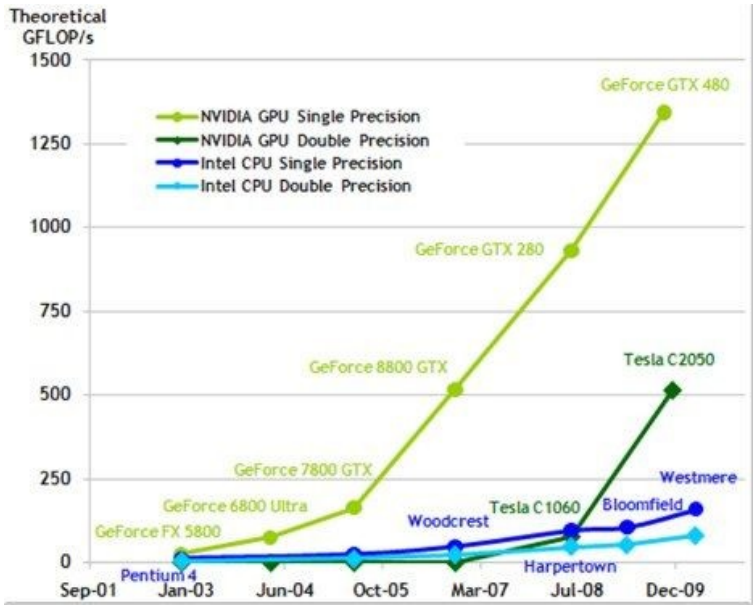
Díky značnému potenciálu a rychlému vývoji grafických karet, je možné v dnešní době provádět paralelní výpočty právě na GPU. OpenCL představuje standard pro psaní výpočetních jader, jež mohou uskutečňovat výpočet v heterogenním prostředí (CPU, GPU,…). Výrobci grafických karet přišly s vlastními technologiemi, které usnadňují programování, jako jsou AMD Stream či CUDA.

Z důvodu značné podpory ze strany výrobce, výkonnostních výsledků a přítomnosti grafické karty na testovacím PC, byla pro implementaci vybrána právě technologie poskytována společností nVidia.

6.1 CUDA

CUDA (Compute Unified Device Architecture) je technologie vyvinuta společností nVidia, která umožňuje využívat GPU k náročným paralelním výpočtům, jež mohou být zcela odlišné od původního zaměření karet určených výhradně zpracování 2D či 3D grafiky.

Důvodem vytvoření takovéto architektury je fakt, že procesory grafických karet začaly výkonnostně značně předhánět klasické procesory. Na následujícím grafu je jasně patrné, jak významně grafické karty nVidia předčí procesory Intel:

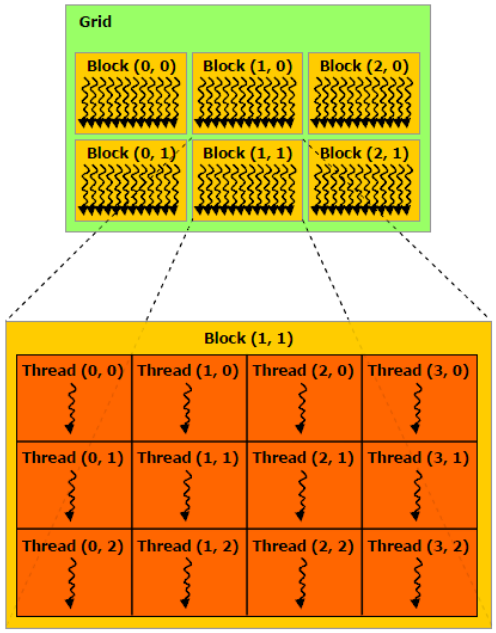


Obrázek 6-1 Porovnání výpočetního výkonu nVidia GPU a Intel CPU [7]

CUDA využívá pro programování například jazyk C for CUDA, což je jazyk C s určitými úpravami a rozšířeními o některé prvky, které existují v jazyce C++ (práce s přetíženými operátory, jmennými prostory, šablonami,…) a navíc o části, jež zavedla společnost nVidia. Naopak tento jazyk má určitá omezení, např.: nemožnost použít rekurzivní funkce na straně GPU.

V květnu 2011 byla vydána CUDA v4.0, která je použita i pro implementaci přiloženého programu. Tuto technologii je možné použít pod operačními systémy Windows XP a vyšší, Mac OS X či LINUX. Pro implementaci aplikací je navíc třeba v prvé řadě nainstalovat příslušné CUDA SDK (Software Development Kit) obsahující potřebná rozhraní včetně kompilátoru nvcc, dále CUDA Developer Driver (driver grafické karty, který podporuje CUDu) a CUDA Toolkit.

Z hlediska architektury je nejvyšším prvkem mřížka, ve které jsou organizovány bloky. Každý blok pracuje nezávislé na ostatních blocích a obsahují určité množství vláken. Vlákna nacházející se ve stejném bloku mohou být synchronizována a sdílet data ve společné sdílené paměti. Velikosti mřížek a bloků jsou závislé na použité grafické kartě.



Obrázek 6-2 CUDA - vláknová architektura [7]

Grafická karta obsahuje několik druhů paměti, které se liší nejen velikostí, ale také přístupem a především rychlostí.

Paměť	Přístup	Viditelnost	Životnost
Paměť textur	R	Všechna vlákna a host	Do uvolnění
Paměť konstant	R	Všechna vlákna a host	Do uvolnění
Globální paměť	R/W	Všechna vlákna a host	Do uvolnění
Sdílená paměť	R/W	Vlákna v bloku	Blok
Lokální paměť	R/W	Vlákno	Vlákno
Paměť registrů	R/W	Vlákno	Vlákno

Tabulka 6-1 Typy pamětí

Základem každého programu využívajícího CUDu je funkce definována klíčovým slovem __global__ , čili tzv. kernel spouštěný N-krát v každém vlákně, které je určeno svým identifikačním číslem. Na obrázku 6-2 je patrná architektura vláken.

Jednoduchost programování paralelních výpočtů je nejlépe viditelná na velice jednoduchém příkladu. Úkolem bude sečíst číselné hodnoty na příslušných místech v poli a výsledek uložit do třetího pole.

```
void ComputeAdd(int threadIdx, int threadCount, int* a, int* b, int*c)

{
   int tid = threadIdx;
   while(tid < N)
   {
      c[i] = a[i] + b[i];
      tid += threadCount;
   }
}
```

Výpis 6-1 Sečtení čísel pomocí vláken na CPU

Budou-li k dispozici pouze dvě vlákna, kdy jejich indexy budou 0 a 1, lze vidět, že jedno vlákno bude sčítat prvky na sudých pozicích. Druhé vlákno s indexem jedna pak hodnoty na lichých pozicích.

Stejný problém je možné realizovat také na GPU. Pomocí technologie CUDA by kód kernelu, který provádí stejný výpočet, mohl vypadat jako na následujícím výpisu.

```
__global__ void kernel(int* a, int* b, int* c)

{
   int tid = threadIdx.x + blockIdx.x * blockDim.x;
   while(tid<N)
   {
      c[tid] = a[tid] + b[tid];
      tid += blockDim.x * gridDim.x;
   }
}
```

Výpis 6-2 Sečtení čísel pomocí GPU

Zjištění indexu pole je dáno architekturou popsanou výše. Index prvku v pole tak získáme z indexu vlákna, bloku a velikosti dimenze. V případě, že velikost pole je ještě větší, musí se index násobit velikostí mřížky v cyklu. Každé vlákno tak sečte prvky ve dvou polích a uloží je do třetího pole.

Velikosti mřížky a bloků jsou zadány na straně aplikace, která spouští kernel. Zde také musí dojít k patřičným alokacím paměti na GPU, zkopírování vstupních dat, získání výsledků a uvolnění paměti.

6.2 Implementace detektoru na GPU

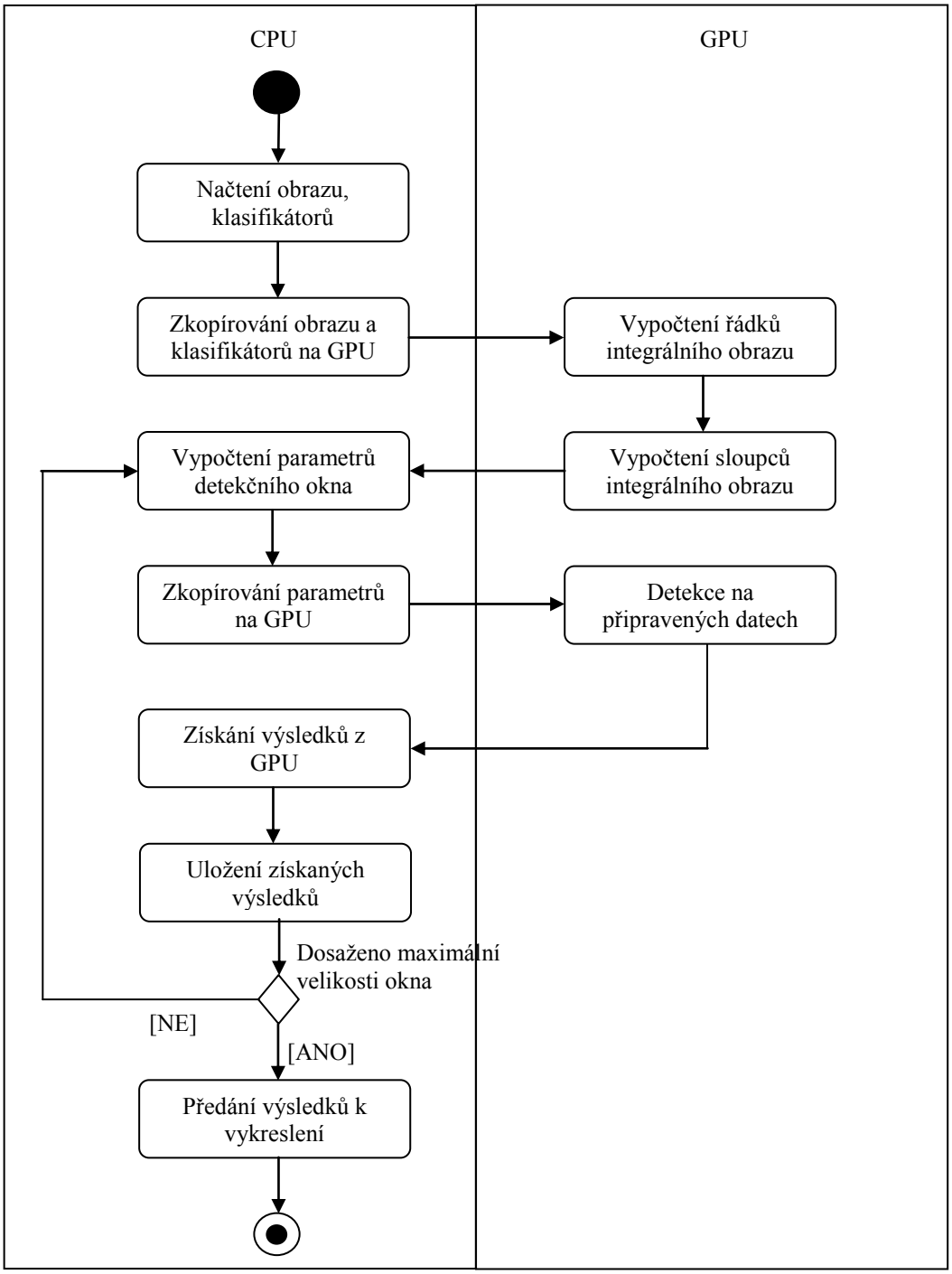
Výpočet integrálního obrazu pomocí GPU lze řešit obdobně, jako pomocí více vláknového přístupu přes CPU. Princip paralelního výpočtu integrálního obrazu je názorně ukázán v kapitole 5. Realizujeme-li danou úlohou pomocí technologie CUDA, je třeba nejdříve alokovat dostatečné místo na zařízení pro uložení obrazových dat. Dále je nutné alokovat dostatečný prostor pro výsledný integrální obraz a integrální obraz druhých mocnin, přičemž je nutné nezapomínat na zvětšení jejich dimenzí o jeden sloupec a řádek oproti původnímu obrazu. Jakmile máme k dispozici dostatečný prostor, dojde k přesunutí dat vstupního obrazu do globální paměti grafické karty.

Přestože v kapitole testování uvidíme, že výpočet integrálního obrazu je časově delší při použití GPU, přesto použijeme tuto metodu. Důvodem jsou delší přenosy dat mezi aplikací a grafickou kartou. Detektor ale nebude muset výsledné integrální obrazy vracet zpět do aplikace, ale naopak je zanechá v paměti grafické karty. Takto se nám sníží přesuny dat, protože v případě výpočtu integrálních obrazů na straně hostitelské aplikace, bylo by nutné oba tyto obrazy přesunout do grafické karty.

Počet vláken ke spuštění výpočtu integrálního obrazu můžeme stanovit na základě počtu dostupných vláken, které poskytuje grafická karta. Velikost mřížky pak můžeme určit jako podíl rozměru obrázku a počet vláken.

Nyní je možné spustit samotný výpočet, který je rozdělen do dvou samostatných kernelů. První z nich vypočítá hodnoty po řádcích. Jakmile svůj úkol dokončí, je spuštěn druhý kernel, jenž dopočítá hodnoty po sloupcích. Integrální obrazy jsou ponechány v globální paměti grafické karty, zatímco paměť alokována pro původní obraz je uvolněna.

Proces detekce řešený pomocí GPU je znázorněn na obrázku 6-3. Celý postup je podobný schématu výpočtu integrálního obrazu.



Obrázek 6-3 Kroky detektoru s využitím GPU

Po získání vstupního obrazu a množiny natrénovaných klasifikátorů, se nastaví velikost mřížek, počet vláken pro hlavní kernel. Implementovaná aplikace používá maximální počet dostupných vláken, který ovlivňuje spolu s velikostí obrazu, aktuálního zvětšení a detekčních oken počet použitých bloků.

$$počet bloků X = \frac{\left(\frac{šířka\ obrazu}{zvětšení\ okna}\right)}{\sqrt{počet\ vláken}}, počet bloků Y = \frac{\left(\frac{výška\ obrazu}{zvětšení\ okna}\right)}{\sqrt{počet\ vláken}}$$

$$počet bloků = počet bloků X \cdot počet bloků Y$$

Pokud budeme mít jako vstup obraz o velikosti 144 x 192, získáme pro zvyšující se velikost detekčních oken, počty bloků uvedené v tabulce 6-2.

Spuštění kernelu	Koeficient velikosti	Počet bloků
1	1,2	24
2	1,44	24
3	1,72	24
4	2,07	6
5	2,48	6
6	2,98	6
7	3,58	2
8	4,29	1
9	5,16	1
10	6,19	1

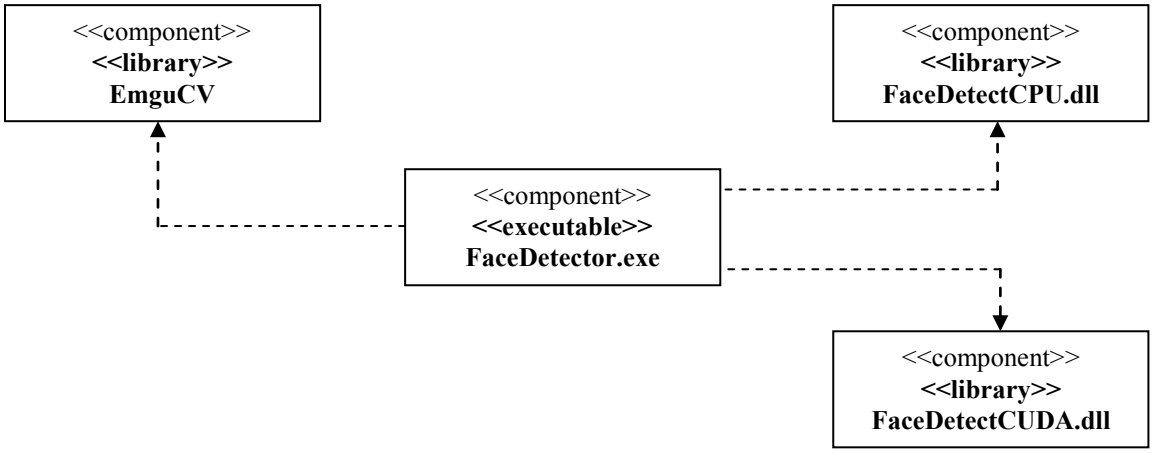
Tabulka 6-2 Počty bloků podle velikosti detekčního okna

Potom proběhne alokace datových struktur na straně klientské aplikace. Potřebné místo pro data, která jsou nutná znát v době výpočtu, se získá i na straně GPU.

Pro aktuální velikostní koeficient se zaktualizuje velikost a maximální možná pozice detekčních oken, tedy informace nutné pro výpočet pozice detekčního okna. Jakmile jsou známy tyto informace, spustí se výpočet detekce. Po ukončení se vrátí výsledek, a pokud jsou ještě nějaká neotestovaná okna, zaktualizují se znovu informace o nich a spustí se další výpočet.

6.3 Výsledná aplikace

Výsledná aplikace je rozdělená do několika částí. Vlastní detektory byly realizovány v programovacím jazyce C++ a z výsledků byly vytvořeny dvě samostatné DLL knihovny, které jsou využívány spustitelnou aplikací implementovanou v jazyce C#.



Obrázek 6-4 Diagram komponent výsledné aplikace

První knihovna FaceDetectCPU.dll obsahuje detektor pracující pomocí CPU a druhá FaceDetectGPU.dll detektor využívající GPU. Obě knihovny poskytují spustitelné aplikaci rozhraní, kterým přijímají vstupy od uživatele anebo mu poskytují výsledky výpočtů.

Dále byla použita volně dostupná knihovna EmguCV [15], která zpřístupňuje funkce knihovny OpenCV [16] v prostředí jazyka C#. Důvodem jejího použití bylo jednoduchost práce s webkamerou, načítání a podpora grafických formátů, převádění obrazu do odstínu šedi a také pro závěrečné porovnání detektorů.

7. Testování

Závěrečné testování bylo zaměřeno především na výkonnost a rychlost detekce, jak pro vstupní obrazy různých velikostí, tak také pro vstup z webkamery. Hlavním cílem práce nebylo vytvořit co možná nejlepší detektor, proto byl trénován z důvodu časové náročnosti jen na menších množinách pro učení. Nicméně bylo nutné nechat natrénovat dostatečný počet klasifikátorů, jejichž počet by odpovídal počtům klasifikátorů kvalitněji naučených detektorů. Pokud by počty příznaků byly nízké, mohlo by to vést k nepřesným výsledkům výkonu, neboť délka výpočtu při detekci je ovlivněna také počtem klasifikátorů.

Po ověření funkčnosti detektoru a zjištění jeho úspěšnosti, se přešlo k hlavnímu výkonnostnímu testování. Testovala se nejenom samotná detekce, ale i výpočet integrálního obrazu. Výsledné hodnoty se poté porovnaly s různými verzemi implementovaného detektoru. Pro analýzu verzí využívající GPU byla použita utilita nVidia Visual Profiler dodávána k CUDA SDK, která obsahuje řadu funkcí a sledovat celou řadu parametrů, například:

- Doba kopírování dat do grafické karty
- Doba kopírování dat z grafické karty
- Doba běhu výpočtu na jednotlivých kernelech
- Grafické znázornění výsledků pomocí grafů
- Porovnávání výsledků
- Množství použitých registrů
- Velikost využité globální paměti
- Velikost využité sdílené paměti
- Počet použitých registrů

Veškeré výkonnostní testy byly prováděny na PC sestavě obsahující procesor Intel Core i5-760 2,8 GHz, 8 GB RAM. Dále pro testování výkonnosti GPU akcelerace byla použita karta nVidia GeForce GTX560Ti, jejichž parametry týkající se technologie CUDA jsou uvedeny v následující tabulce 7-1.

Vlastnost	Hodnota	
Název	GeForce GTX 560 Ti	
Compute capability (verze architektury CUDA na daném hardware)	2.1	
Počet multiprocesorů	8	
Velikost globální paměti	1073741824 bajtů	
Velikost paměti pro konstanty	65536 bajtů	
Velikost sdílené paměti pro multiprocesor	49152 bajtů	
Počet registrů pro multiprocesor	32768	
Maximální velikost mřížky	X	65535
	Y	65535
	Z	65535
Maximální velikost bloků	X	1024
	Y	1024
	Z	64
Maximální počet vláken v bloku	1024	
Počet vláken ve warpu (warp je skupina vláken zpracovávaných současně	32	

Tabulka 7-1 CUDA parametry grafické karty

7.1 Testování úspěšnosti detektoru

Pro hlavní výkonnostní testování bylo postupně natrénováno několik množin klasifikátorů. Databáze obličejů, kterou poskytuje MIT CBCL obsahuje krom vzorků pro učení, také množinu obrazu pro testování úspěšnosti připraveného detektoru. Z této množiny byl vybrán jeden vhodný

obraz obsahující 57 obličejů, který bude následně používán pro otestování úspěšnosti natrénovaných detektorů.

Před zjišťováním úspěšnosti detektorů je třeba definovat veličiny, které budou v měření vystupovat:

True Positive (TP) – objekt patří do pozitivní třídy a je klasifikován jako pozitivní
False Negative (FN) – objekt patří do pozitivní třídy a je klasifikován jako negativní
True Negative (TN) – objekt patří do negativní třídy a je klasifikován jako negativní
False Positive (FP) – objekt patří do negativní třídy a je klasifikován jako pozitivní
True Positive Rate (TPR) – je dán jako podíl nalezených objektů patřící do hledané třídy se všemi hledanými objekty na obrazu (součet počtu nalezených objektů s nenalezenými hledanými objekty)

$$TPR = \frac{TP}{TP + FN}.$$

False Positive Rate (FPR) – je dán jako podíl nalezených objektů, které nepatří do hledané třídy se všemi nalezenými okny na obrazu (součet nalezených hledaných objektů se špatně nalezenými objekty)

$$FPR = \frac{FP}{TP + FP}.$$

- Celková úspěšnost (A_{cc}) – je pak dán jako

$$A_{cc} = \frac{TP + TN}{TP + TN + FP + FN}.$$

V první fázi byl natrénován monolitický detektor, tedy detektor s jedním silným klasifikátorem. Obsahuje 200 slabých klasifikátorů, které budou jednotlivě aplikovány na detekční okno. Dále jsou připraveny dvě množiny kaskád. Oba mají stejný počet stupňů kaskády, ale rozlišují je počty použitých vzorků k učení. Ve výsledku mají i kaskádové detektory různý počet natrénovaných klasifikátorů, aby se ověřilo, jak jejich počet ovlivní výslednou rychlost detekce.

Pro všechny detekční testy bylo použito stejné nastavení parametrů pro detekci, které je uvedeno v tabulce 7-2.

Parametr	Hodnota
Koeficient zvětšení:	1,2
Šířka minimálního okna	23 pixelů
Výška minimálního okna	23 pixelů

Tabulka 7-2 Parametry detektoru

Hodnoty pro testování uvedené v tabulce výše popisují, že detekční okna, se kterými detektor začíná pracovat, bude mít rozměry 23x23 pixelů. Po dosažení posledního okna se rozměry okna a příznaků vynásobí koeficientem 1,2. Tento koeficient říká, že se rozměry budou vždy zvětšovat o 20% oproti předchozímu stavu.

Vstupní parametry:

Parametr	Hodnota
Databáze	MIT CBCL
Počet slabých klasifikátorů	200
Počet pozitivních vzorů	2429
Počet negativních vzorů	6636
Šířka vzorku	19 pixelů
Výška vzorku	19 pixelů

Tabulka 7-3 Vstupní parametry monolitického detektoru

Pro monolitický detektor bylo natrénováno:

Silný klasifikátor	Počet slabých klasifikátorů
1	200
Celkem	**200**

Tabulka 7-4 Výstup trénování monolitického detektoru

Výsledek:

Obrázek 7-1 Výsledky detekce monolitického detektoru

Vstupní parametry:

Parametr	Hodnota
Databáze	MIT CBCL
Počet stupňů kaskády	16
Detection rate	0,999
False positive rate	0,5
Počet pozitivních vzorů	1600
Počet negativních vzorů	1600
Šířka vzorku	19 pixelů
Výška vzorku	19 pixelů

Tabulka 7-5 Vstupní parametry kaskádového detektoru 1

Pro kaskádový detektor bylo natrénováno:

Stupeň kaskády	Počet slabých klasifikátorů	Stupeň kaskády	Počet slabých klasifikátorů
1.	3	9.	20
2.	5	10.	23
3.	6	11.	25
4.	11	12.	25
5.	11	13.	27
6.	14	14.	27
7.	17	15.	27
8.	17	**Celkem**	**258**

Tabulka 7-6 Výstup trénování kaskádového detektoru 1

Výsledek:

Obrázek 7-2 Výsledky detekce kaskádového detektoru 1

Vstupní parametry:

Parametr	Hodnota
Databáze	MIT CBCL
Počet stupňů kaskády	15
Detection rate	0,999
False positive rate	0,5
Počet pozitivních vzorů	2900
Počet negativních vzorů	4200
Šířka vzorku	19 pixelů
Výška vzorku	19 pixelů

Tabulka 7-7 Vstupní parametry kaskádového detektoru 2

Pro kaskádový detektor bylo natrénováno:

Stupeň kaskády	Počet slabých klasifikátorů	Stupeň kaskády	Počet slabých klasifikátorů
1.	12	9.	85
2.	16	10.	95
3.	24	11.	95
4.	35	12.	100
5.	48	13.	102
6.	57	14.	103
7.	73	15.	90
8.	74	**Celkem**	**1009**

Tabulka 7-8 Výstup trénování kaskádového detektoru 2

Výsledek:

Obrázek 7-3 Výsledky detekce kaskádového detektoru 2

Ve výstupních obrazech jsou zeleně označeny správně rozpoznané obličeje. Modře jsou označené ty obličeje, které nebyly detektorem rozpoznány a červeně jsou zvýrazněny oblasti, které byly detektorem rozpoznány jako obličeje, ale nejedná se o ně.

Konečné výsledky a porovnání připravených detektorů pro testovaný obraz je zaznamenán v tabulce 7-9. Dle výsledků je patrný především rozdíl mezi kaskádovými detektory. Přestože mají oba dva stejný počet stupňů kaskády, druhý má dle výsledků lepší detekční schopnost. Je tomu především díky větší množině pozitivních a negativních vzorků:

Detektor	Správně označené	Špatně označené	Celková úspěšnost
Monolitický detektor	39	2	66 %
Kaskádový detektor 1	27	1	46 %
Kaskádový detektor 2	53	2	86 %

Tabulka 7-9 Výsledná úspěšnost detektorů

7.2 Testování výkonu učícího programu

Pro trénování byla použita více vláknová varianta učícího algoritmu, která je detailněji popsána v kapitole 4, kdy byly zjištěny hodnoty délky určení jednoho slabého klasifikátoru, které jsou uvedeny v tabulce 7-10.

Detektor	1 vlákno	2 vlákna	4 vlákna
Monolitický detektor	1020 s	540 s	290 s
Kaskádový detektor 1	400 s	220 s	130 s
Kaskádový detektor 2	760 s	390 s	200 s

Tabulka 7-10 Výsledky výkonnosti trénování klasifikátorů

Z výsledků je vidět, že přidáváním dalších procesorových jader, se doba výpočtu zkracuje a to přibližně tolikrát, kolik jader je použito. Přidáním dalších výpočetních jednotek by došlo k dalšímu zkrácení doby výpočtu.

Nicméně tyto výsledky jsou uvedeny pro výpočet pouze jednoho slabého klasifikátoru. Pro monolitický klasifikátor bylo natrénováno 200 takovýchto klasifikátorů. Vezmeme-li nejlepší výkonnostní výsledek, jehož bylo dosaženo nasazením 4 procesorových jader, zjistíme, že doba výpočtu monolitického detektoru se přibližně rovná 16 hodin. Analogicky dále lze určit, že doba výpočtu kaskádového detektoru 1 trvala 10 hodin a kaskádového detektoru 2 přibližně 57 hodin.

7.3 Testování výkonu výpočtu integrálního obrazu

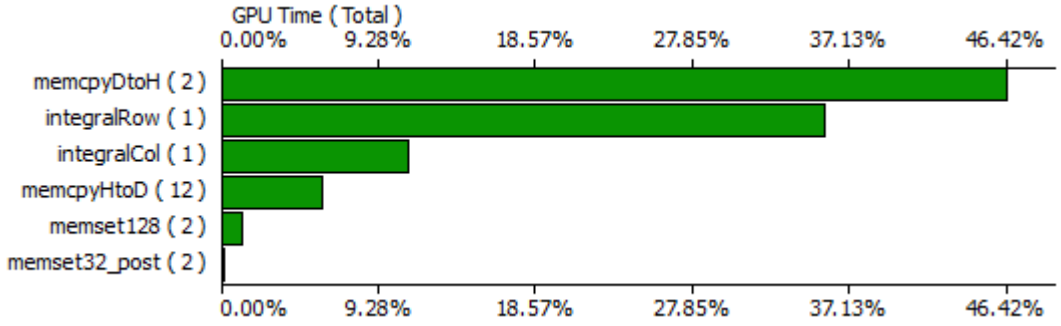
Jak bylo ukázáno v kapitole 5, je možné paralelně vypočítávat také integrální obrazy. Byly provedeny jak implementace bez použití vláken, tak zároveň více vláknové varianty běžící na CPU i GPU.

Velikost obrazu	1 vlákno	4 vlákna	GPU
716 x 684	0,004 s	0,01 s	0,042 s
1280 x 1024	0,013 s	0,02 s	0,052 s
2250 x 2250	0,056 s	0,07 s	0,072 s
5390 x 7340	0,429 s	0,175 s	0,16 s

Tabulka 7-11 Výsledky výkonnosti výpočtu integrálního obrazu

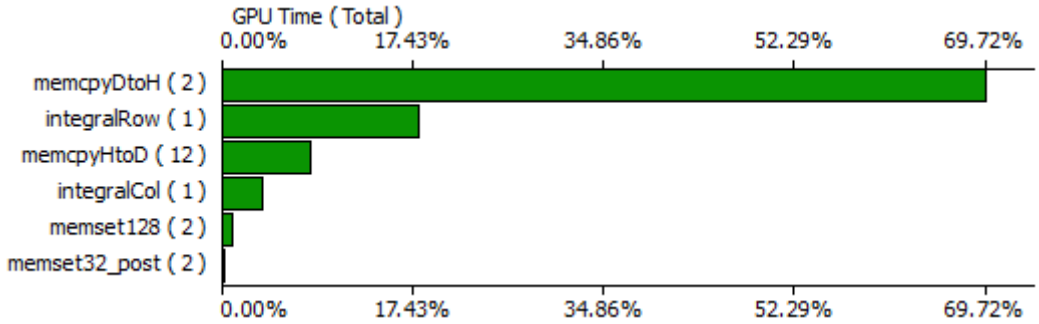
Z výsledků vyplývá, že nechat vypočítávat integrální obrazy na GPU se nehodí v případě menších vstupů. Až u obrazů s velkými rozměry dojde k situaci, kdy výpočet přes GPU je rychlejší.

Důvod proč tomu tak je, osvětluje graf vytvořený za použití utility nVidia Compute Visual Profiler, který poskytuje ucelený pohled na zatížení jednotlivých operací.



Graf 7-1 Graf zatížení GPU při výpočtu integrálního obrazu pro obraz 716 x 684

Z výše uvedeného grafu je patrné, že samotný výpočet, který představují kernely integralRow a integralCol, je méně časově náročný, než přenos obrazových dat ze zařízení do aplikace. Také přesouvání dat opačným směrem je poměrně náročné, což v konečném důsledku znamená, že celý výpočet integrálního obrazu je pro menší obrazy časově náročnější.



Graf 7-2 Graf zatížení GPU při výpočtu integrálního obrazu pro obraz 2250 x 2250

Pro větší obrazy zůstává přesouvání dat stále nejnáročnější operací, ale se vzrůstajícími rozměry dochází k prodlužení výpočtů i na CPU. Od velikosti obrazu 2250x2250 se doba výpočtu přes GPU vyrovnává době výpočtu na CPU.

Dalo by se z uvedených výpočtů tvrdit, že pro menší obrazy nemá smysl nechat vypočítávat integrální obrazy GPU. Je však nutné si uvědomit, že v případě výpočtů na straně CPU, by bylo nutné přesouvat oba vytvořené integrální obrazy do globální paměti grafické karty, což znamená časovou ztrátu. Jelikož však bude zapotřebí integrálních obrazů při detekci, je lepší nechat výpočet na GPU, protože takto přesuneme pouze obrazová data, vypočteme obrazy a ty necháme v globální paměti a můžeme s nimi dále pracovat. Nebude tedy nutné je již kopírovat.

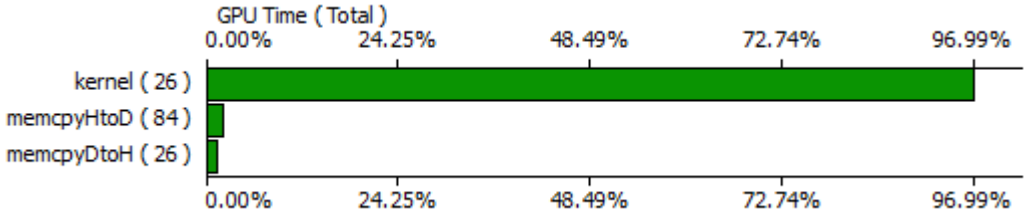
7.4 Porovnání výkonu monolitického detektoru

První byl otestován monolitický detektor obsahující 200 slabých klasifikátorů. V tabulce 7-12 jsou výsledky pro tři různě velké obrazy, na kterých byla otestována jedno-vláknová, více-vláknová a GPU varianta detektoru.

Velikost obrazu	1 vlákno	4 vlákna	GPU
716 x 684	14,664 s	10,294 s	0,214 s
1280 x 1024	55,833 s	38,200 s	0,526 s
2250 x 2250	337,805 s	180,792 s	2,084 s

Tabulka 7-12 Výsledky výkonnosti detekce monolitického detektoru

Zatížení jednotlivých operací při výpočtu přes GPU je zaznamenáno v grafu 7-3, kde je patrné, že samotný výpočet je nejnáročnější částí. Přenosy dat již nemají tak značný vliv na celkovou dobu výpočtu, jak tomu bylo v případě vytváření integrálních obrazů.



Graf 7-3 Graf zatížení GPU při detekci monolitickým detektorem

Ze zjištěných výsledků dochází ke značnému zrychlení při výpočtu přes GPU. S rostoucí velikostí vstupního obrazu je rozdíl v rychlosti výpočtu stále větší. Zatímco u obrazu 716 x 684 je GPU detektor pouze 49krát rychlejší, u obrazu o velikosti 2250 x 2250 je to už 160krát.

7.5 Porovnání výkonu kaskádového detektoru

V druhé fázi výkonnostního testování se přešlo k připraveným kaskádovým detektorům. Testování proběhlo na stejně velkých obrazech jako v případě monolitického detektoru. Z výsledků uvedených v tabulce 7-14 a tabulce 7-15 vyplývá, že kaskádový detektor způsobuje značné urychlení detekčního procesu, aniž by to v případě kaskádového detektoru 2 nepříznivě ovlivnilo úspěšnost detekce.

Hlavním důvodem vyšší rychlosti detekce je právě mechanismus vyhodnocování detekčních oken, který byl přiblížen v kapitole 3.6.

Stupeň kaskády	Počet detekčních oken k testování	Počet testování
1.	37319	447828
2.	13334	213344
3.	3797	91128
4.	848	29680
5.	332	15936
6.	160	9120
7.	82	5986
8.	57	4218
9.	50	4250
10.	41	3895
11.	34	3230
12.	29	2900
13.	17	1734
14.	12	1236
15.	5	450
Výsledek	**3**	

Tabulka 7-13 Snižování počtu detekčních oken

Tabulka 7-13 poukazuje na postupně se snižující počet nalezených obličejů po aplikování každého stupně kaskády. Navíc z ní zjistíme, že během kaskádové detekce dojde celkově k 835835 testování detekčních oken. Přestože kaskádový detektor 2 obsahuje 1009 klasifikátorů, provede se menší počet porovnání, než je tomu v případě monolitického detektoru s 200 klasifikátory. U něj se uskuteční celkem k 7463800 testování, protože každé detekční okno se musí otestovat s každým klasifikátorem.

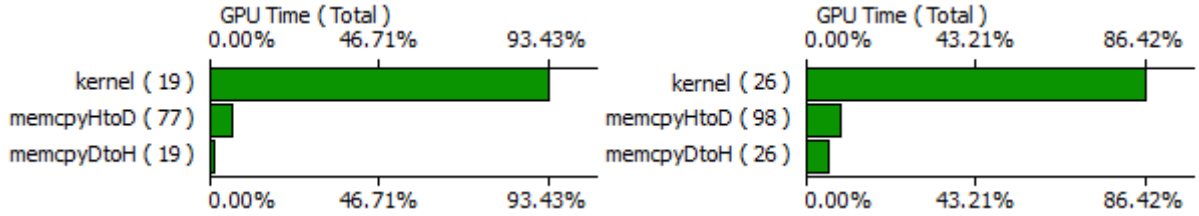
Velikost obrazu	1 vlákno	4 vlákna	GPU
716 x 684	0,452 s	0,16 s	0,092 s
1280 x 1024	1,529 s	0,392 s	0,21 s
2250 x 2250	7,863 s	1,48 s	0,508 s

Tabulka 7-14 Výsledky výkonnosti detekce kaskádového detektoru 1

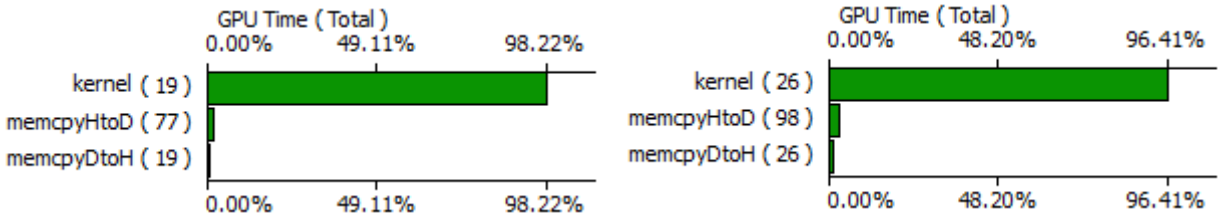
Velikost obrazu	1 vlákno	4 vlákna	GPU
716 x 684	1,482 s	0,474 s	0,1 s
1280 x 1024	7,698 s	1,364 s	0,27 s
2250 x 2250	29,937 s	10,34 s	0,84 s

Tabulka 7-15 Výsledky výkonnosti detekce kaskádového detektoru 2

Z výsledků vidíme, že kaskádový detektor 1 díky menšímu počtu klasifikátorů dosahuje vyššího výkonu. Má však menší detekční schopnost, tedy dochází v každém stupni kaskády k zavrhování většímu počtu oken, což je příčinou vyšší rychlost detekce.

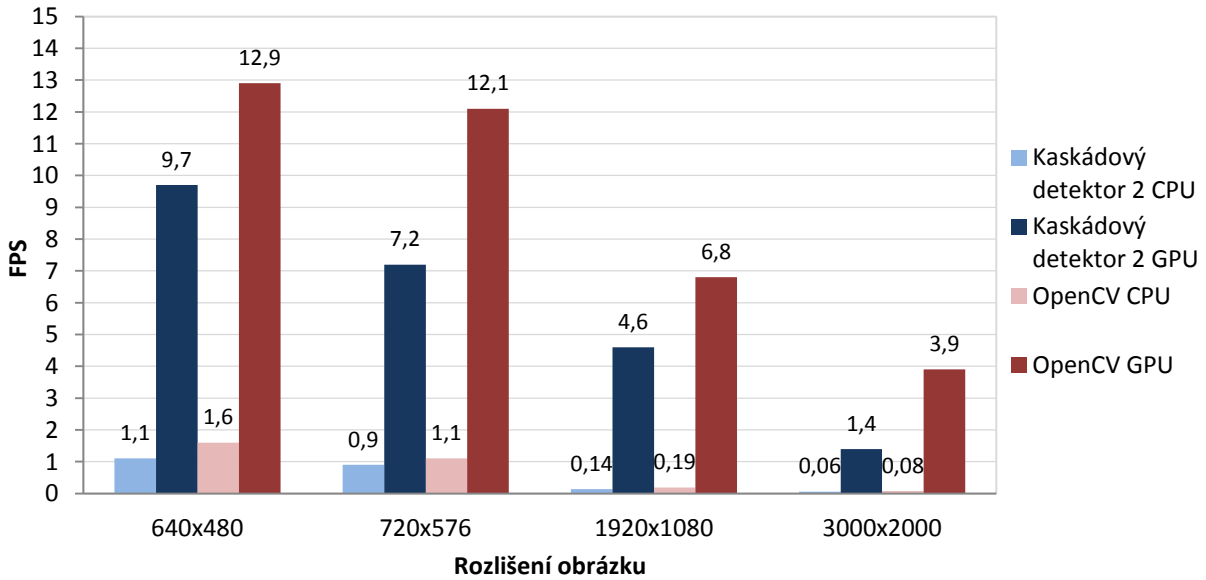


Obrázek 7-4 Vlevo zatížení pro obraz 716x684, vpravo pro 2250x2250 při kaskádovém detektoru 1



Obrázek 7-5 Vlevo zatížení pro obraz 716x684, vpravo pro 2250x2250 při kaskádovém detektoru 2

Po analýze úspěšnosti a výkonnosti vytvořeného detektoru, byl nakonec porovnán s existující verzí detektoru, která je součástí knihovny OpenCV. Detektor používaný v této knihovně využívá pouze CPU, ale je možné s její pomocí provádět detekci pomocí technologie CUDA.



Kritériem porovnání bylo pouze celkový počet detekovaných snímků za sekundu v závislosti na velikosti vstupního obrazu přicházejícího z kamery.

8. Závěr

Prvním krokem při zpracování této práce bylo popsat metody automatické detekce obličejů v obraze. Jelikož existuje celá řada postupů, které řeší detekci různými způsoby, jsou jednotlivé metody rozděleny do základních skupin. Principy několika významnějších z nich jsou zde následně také teoreticky popsány.

Pro praktickou část byl z existujících metod vybrán algoritmus Viola a Jones, který byl použit i pro závěrečné testování a porovnání. Nejprve byly shrnuty teoretické poznatky, se kterými bylo nutné se seznámit před započetím implementace. Poté již bylo možné přejít k samotnému vývoji praktické části.

Výsledný program byl rozdělen do dvou samostatných aplikací: učitele a detektoru. Posláním programu pro trénování je vytvořit soubor dat obsahující množinu natrénovaných klasifikátorů, který je posléze využíván detekčním programem pro nalezení obličejů ve vstupním obraze. V prvním kroku byla připravena verze detektoru provádějící výpočet klasickou cestou pomocí procesoru. Teprve poté byla tato verze upravena tak, aby byl detekční proces prováděn přes grafickou kartu pomocí technologie CUDA.

Jakmile byly připraveny detektory, bylo učitelem natrénováno několik množin klasifikátorů, aby bylo možné zjistit, jak jejich množství ovlivní nejen detekční schopnost ale především výkonnost detektoru. Z tohoto důvodu byla také záměrně připravena množina naučená na menší skupině vzorků obsahující ve výsledku malý počet klasifikátorů, u které se neočekávala velká úspěšnost detekce.

Po ukončení procesu učení, bylo možné nacvičené množiny otestovat. Před výkonnostním testováním byla nejprve zjišťována úspěšnost detektoru na všech připravených množinách. Poté byly porovnány CPU a GPU verze detektorů. Z výsledků je zjevné, že při přenesení výpočtu na grafickou kartu došlo ke značnému urychlení, které bylo znatelné především při zpracování obrazu z webkamery. Verze určená pro procesor dosahovala rychlosti pouze 0,8 FPS, zatímco verze používající technologii CUDA dosahovala rychlosti až 10 FPS.

Během testování bylo také zjištěno, že ne pro všechny druhý výpočtů je vždy vhodné užití grafických karet. Například pro operaci výpočtu integrálního obrazu se pro obrazy menší než 2250x2250 pixelů tento způsob výpočtů naopak nehodí. Nicméně ze zjištěných výsledků vyplývá, že podstatně výkonnostně náročnější detekce obličejů je vhodný proces, který je možné urychlit právě využitím rychlých výpočtů grafických karet.

9. Reference

[1] VIOLA, Paul a Michael JONES. Robust Real-time Object Detection. *International Journal of Computer Vision*. 2004, č. 57, s. 137-154. Dostupné z: http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_IJCV.pdf

[2] YANG, Ming-Hsuan, David J. KRIEGMAN a Narendra AHUJA. Detecting Faces in Images: A Survey. *IEEE Transactions on pattern analysis and machine intelligence*. 2002, č. 24, s. 34-58. ISSN 0162-8828 Dostupné z: http://vision.ucsd.edu/kriegman-grp/papers/pami02.pdf

[3] CHEN, Yao-Jiunn, Yen-Chun LIN. Simple Face-detection Algorithm Based on Minimum Facial Features. *The 33rd Annual Conference of the IEEE Industrial Electronics Society*. 2007, s. 455-460. ISSN 1553-572X. Dostupné z: http://cat.hfu.edu.tw/~b8403009/TD-006386.pdf

[4] TRIPATHI, Smita, Varsha SHARMA a Sanjeev SHARMA. Face Detection using Combined Skin Color Detector and Template Matching Method. *International Journal of Computer Applications*. 2011, s. 5-8. Dostupné z: http://www.ijcaonline.org/volume26/number7/pxc3874290.pdf

[5] SINHA, Pawan. Qualitative Representations for Recognition. *Lecture Notes In Computer Science*. 2002, s. 249-262. Dostupné z: http://web.mit.edu/bcs/sinha/papers/qualitative_reps.pdf

[6] KRUEGER, Jon, Marshall ROBINSON, Doug KOCHELEK a Matthew ESCARRA. Obtaining the Eigenface Basis. In: [online]. [cit. 2012-03-13]. Dostupné z: http://cnx.org/content/m12531/latest/

[7] NVIDIA CORPORATION. *NVIDIA CUDA C Programming Guide*: *Version 4.0* [online]. 2011 [cit. 2012-03-13]. Dostupné z: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

[8] SMACH, F., M. ATRI, J. MITÉRAN a M. ABID. Design of a Neural Networks Classifier. [online]. 2005, s. 124-127 [cit. 2012-03-13]. Dostupné z: http://www.waset.org/journals/waset/v11/v11-31.pdf

[9] CHO, Junguk, Shahnam MIRZAEI, Jason OBERG a Ryan KASTNER. FPGA-Based Face Detection System. *Field Programmable Logic and Applications*. 2008, s. 373-378. Dostupné z: http://cseweb.ucsd.edu/~kastner/papers/fpga09-face_detection.pdf

[10] PHAM, Minh-Tri a Tat-Jen PHAM. Fast training and selection of Haar features using statistics in boosting-based. *Proceedings of the International Conference on Computer Vision*. 2007, s. 1-7. ISSN 1550-5499. Dostupné z: http://web.mysites.ntu.edu.sg/astjcham/public/Shared%20Documents/papers/PhamCham-ICCV07.pdf

[11] EYEDEA RECOGNITION. *Advanced computer vision solutions* [online]. [cit. 2012-04-12]. Dostupné z: http://www.eyedea.cz/

[12] CBCL SOFTWARE. [online]. [cit. 2012-04-12]. Dostupné z: http://cbcl.mit.edu/software-datasets/FaceData2.html

[13] Obtaining the Eigenface Basis. [online]. [cit. 2012-04-12]. Dostupné z: http://cnx.org/content/m12531/latest/

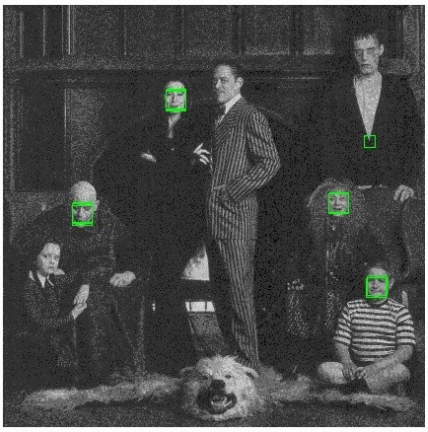
[14] ROWLEY, Henry A., Shumeet BALUJA a Takeo KANADE. Neural Network face detection. 1998, s. 23-38. ISSN 0162-8828. Dostupné z: http://www.informedia.cs.cmu.edu/documents/rowley-ieee.pdf

[15] *EmguCV* [online]. [cit. 2012-04-14]. Dostupné z: http://www.emgu.com/wiki/index.php/

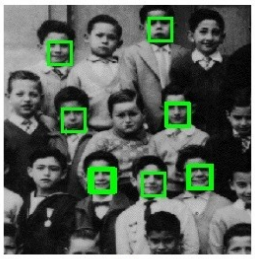
[16] *OpenCV* [online]. [cit. 2012-04-14]. Dostupné z: http://opencv.willowgarage.com/wiki/

A. Ukázka detekce tváří na dalších obrázcích

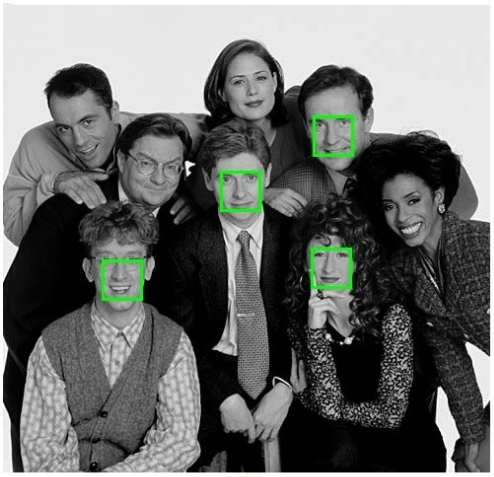
Monolitický klasifikátor



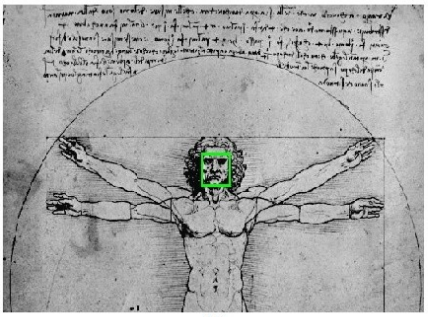
(1)



(2)



(3)

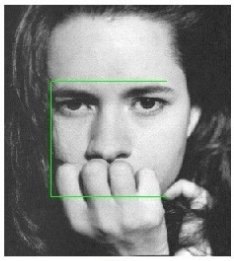


(4)

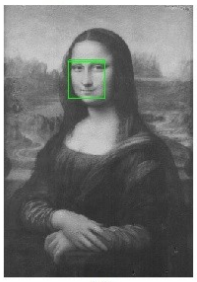
(5)

(9)

(6)



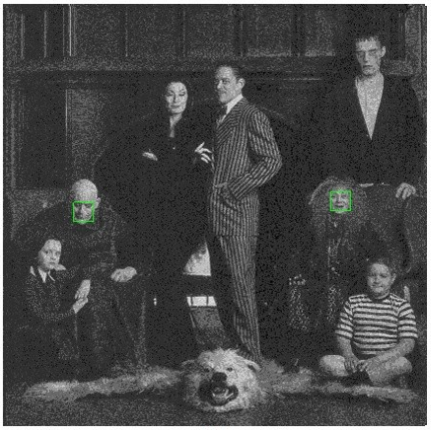
(7)



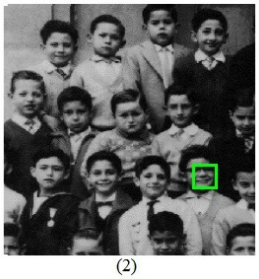
(8)

#	velikost obrazu	obličejů	správně označené	špatně označené	čas GPU detekce
1	864x890	6	4	1	0,296 s
2	256x256	18	7	0	0,032 s
3	500x500	8	4	0	0,108 s
4	592x843	1	1	0	0,188 s
5	280x484	1	0	1	0,064 s
6	340x350	6	3	0	0,06 s
7	592x654	1	1	0	0,16 s
8	520x739	1	1	0	0,156 s
9	500x500	6	2	0	0,108 s

Kaskádový detektor 1

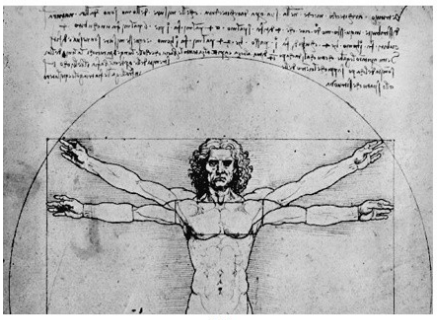


(1)



(2)

(3)

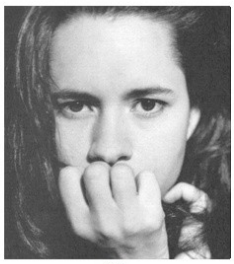


(4)

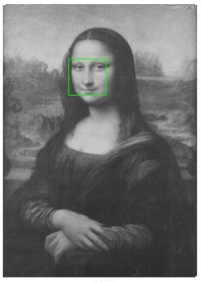
(5)

(9)

(6)



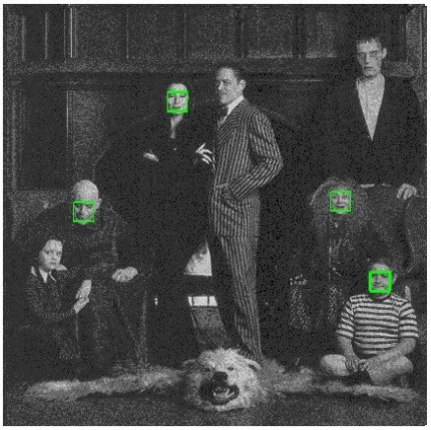
(7)



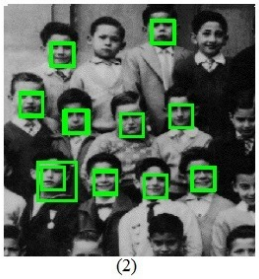
(8)

#	velikost obrazu	obličejů	správně označené	špatně označené	čas GPU detekce
1	864x890	6	2	0	0,08 s
2	256x256	18	1	0	0,032 s
3	500x500	8	1	0	0,048 s
4	592x843	1	0	0	0,06 s
5	280x484	1	0	0	0,032 s
6	340x350	6	1	0	0,028 s
7	592x654	1	0	0	0,06 s
8	520x739	1	1	0	0,06 s
9	500x500	6	0	0	0,048 s

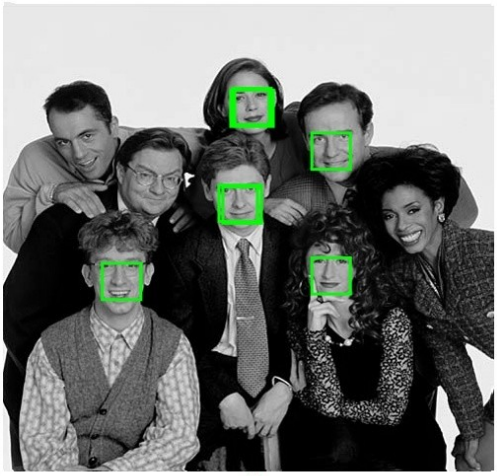
Kaskádový detektor 2



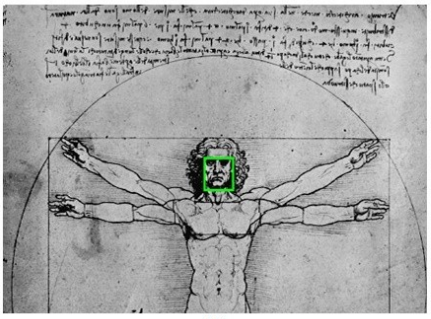
(1)



(2)



(3)

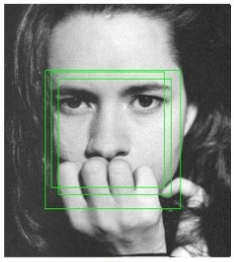


(4)

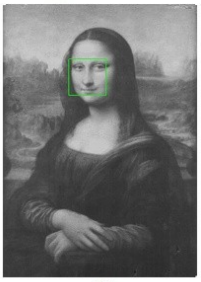
(5)

(9)

(6)



(7)



(8)

#	velikost obrazu	obličejů	správně označené	špatně označené	čas GPU detekce
1	864x890	6	4	0	0,156 s
2	256x256	18	10	1	0,032 s
3	500x500	8	5	0	0,064 s
4	592x843	1	1	0	0,096 s
5	280x484	1	1	0	0,028 s
6	340x350	6	3	0	0,048 s
7	592x654	1	1	0	0,096 s
8	520x739	1	1	0	0,096 s
9	500x500	6	5	0	0,064 s

#	Rozměr	Počet obličejů	CPU											
			Monolitický detektor 1			Kaskádový detektor 1			Kaskádový detektor 2			OpenCV Detektor		
			Spravné	Chybné	Čas [s]	Spravné	Chybné	Čas [s]	Spravné	Chybné	Čas [s]	Spravné	Chybné	Čas [s]
1	144x192	1	1	0	0,5	0	0	0,064	1	0	0,11	1	0	0,094
2	320x214	5	3	0	1,39	1	0	0,06	4	0	0,188	4	0	0,141
3	412x273	1	1	0	2,262	1	0	0,108	1	0	0,28	1	2	0,265
4	306x472	4	0	0	2,98	1	0	0,14	1	0	0,36	3	0	0,265
5	336x484	2	2	0	3,384	2	0	0,156	2	0	0,482	2	0	0,327
6	628x454	11	1	0	6,176	3	0	0,248	4	1	0,842	8	2	0,578
7	800x1024	1	1	3	19,67	1	0	0,795	1	0	2,106	1	0	1,754
8	1280x720	3	2	1	21,31	1	0	0,874	2	0	2,449	3	0	1,529
9	1200x778	11	1	0	21,278	0	0	0,858	4	1	2,403	8	3	1,809
10	1222x864	21	8	1	23,993	6	0	1,016	9	0	2,622	21	2	1,7
11	1280x1024	1	0	0	30,624	0	1	1,186	0	0	2,93	1	3	2,73
12	1500X982	153	59	1	34,133	14	1	1,498	96	1	4,212	151	6	3,104
13	1500x1200	154	30	1	42,432	12	1	1,732	38	1	4,82	65	0	5,188
14	1600x1068	16	0	3	39,557	0	1	1,622	4	5	4,774	8	6	3,717
15	1600x1200	9	5	0	43,945	4	1	1,778	9	0	4,352	9	0	2,09
16	1600x1200	29	2	1	44,866	0	0	1,841	8	1	6,131	21	7	3,994
17	1600x1200	6	6	0	43,774	5	1	1,763	6	0	4,836	6	0	2,153
18	1920x1080	3	1	0	48,329	1	1	1,872	1	0	4,93	3	3	3,51
19	2866x2112	5	1	7	142,772	1	0	5,46	1	1	15,772	5	1	11,796
20	3056x2038	3	3	4	148,466	3	1	6,1	3	3	17,924	3	14	14,57
21	3056x2038	8	2	6	148,809	1	1	6,192	7	4	18,438	7	17	12,98
22	3000x1999	48	11	15	144,161	5	4	5,694	32	3	15,756	46	2	13,275
23	3500x1852	19	1	5	155,018	2	1	6,396	4	5	17,192	18	0	11,808
24	4000x3000	3	0	4	286,948	1	1	11,31	2	2	31,046	3	11	27,566
25	4480x3584	57	57	6	392,094	51	4	16,036	57	11	42,854	55	11	34,001

C. Srovnání GPU

#	Rozměr	Počet obličejů	GPU											
			Monolitický detektor 1			Kaskádový detektor 1			Kaskádový detektor 2			OpenCV Detektor		
			Spravné	Chybné	Čas [s]	Spravné	Chybné	Čas [s]	Spravné	Chybné	Čas [s]	Spravné	Chybné	Čas [s]
1	144x192	1	1	0	0,032	0	0	0,014	1	0	0,016	1	0	0,031
2	320x214	5	3	0	0,032	1	0	0,016	4	0	0,032	4	0	0,016
3	412x273	1	1	0	0,046	1	0	0,016	1	0	0,032	1	2	0,016
4	306x472	4	0	0	0,062	1	0	0,032	1	0	0,03	3	0	0,015
5	336x484	2	2	0	0,062	2	0	0,032	2	0	0,03	2	0	0,031
6	628x454	11	1	0	0,11	3	0	0,046	4	1	0,078	8	2	0,041
7	800x1024	1	1	3	0,296	1	0	0,062	1	0	0,11	1	0	0,073
8	1280x720	3	2	1	0,328	1	0	0,078	2	0	0,125	3	0	0,063
9	1200x778	11	1	0	0,328	0	0	0,078	4	1	0,109	8	3	0,047
10	1222x864	21	8	1	0,374	6	0	0,078	9	0	0,156	21	2	0,077
11	1280x1024	1	0	0	0,437	0	1	0,094	0	0	0,13	1	3	0,094
12	1500X982	153	59	1	0,499	14	1	0,11	96	1	0,187	151	6	0,094
13	1500x1200	154	30	1	0,609	12	1	0,125	38	1	0,198	65	5	0,094
14	1600x1068	16	0	3	0,562	0	1	0,125	4	5	0,218	8	6	0,078
15	1600x1200	9	5	0	0,655	4	1	0,14	9	0	0,191	9	0	0,094
16	1600x1200	29	2	1	0,634	0	0	0,125	8	1	0,265	21	7	0,109
17	1600x1200	6	6	0	0,655	5	1	0,14	6	0	0,203	6	0	0,093
18	1920x1080	3	1	0	0,717	1	1	0,141	1	0	0,234	3	3	0,112
19	2866x2112	5	1	7	1,998	1	0	0,312	1	1	0,592	5	1	0,23
20	3056x2038	3	3	4	2,106	3	1	0,372	3	3	0,702	3	14	0,267
21	3056x2038	8	2	6	2,106	1	1	0,36	7	4	0,732	7	17	0,29
22	3000x1999	48	11	15	2,043	5	4	0,343	32	3	0,6	46	2	0,234
23	3500x1852	19	1	5	2,2	2	1	0,376	4	5	0,672	18	0	0,277
24	4000x3000	3	0	4	4,088	1	1	0,592	2	2	1,122	3	11	0,741
25	4480x3584	57	57	6	5,506	51	4	0,856	57	11	1,652	55	11	0,921

D. Ukázka výstupní ho XML souboru s natrénovanými klasifikátory

```
<Detector>
   <width> 19 </width>
   <height> 19 </height>
   <strongier>
      <weakier>
         <feature>
            <rectangle> 6 1 9 7 -1 </rectangle>
            <rectangle> 9 1 3 7 3 </rectangle>
         </feature>
         <threshold> 21.0103 </threshold>
         <alpha> 2.26734 </alpha>
         <polarity> 0 </polarity>
      </weakier>
      <weakier>
         <feature>
            <rectangle> 1 4 19 6 -1 </rectangle>
            <rectangle> 1 7 19 3 2 </rectangle>
         </feature>
         <threshold> 22.3893 </threshold>
         <alpha> 1.79809 </alpha>
         <polarity> 0 </polarity>
      </weakier>
      <threshold> -6.79082 </threshold>
   </strongier>
   <strongier>
      <weakier>
         <feature>
            <rectangle> 8 3 6 8 -1 </rectangle>
            <rectangle> 10 3 2 8 3 </rectangle>
         </feature>
         <threshold> 14.2428 </threshold>
         <alpha> 1.85502 </alpha>
         <polarity> 0 </polarity>
      </weakier>
      <weakier>
         <feature>
            <rectangle> 15 3 5 15 -1 </rectangle>
            <rectangle> 15 8 5 5 3 </rectangle>
         </feature>
         <threshold> 34.167 </threshold>
         <alpha> 1.6006 </alpha>
         <polarity> 0 </polarity>
      </weakier>
      <threshold> -5.72905 </threshold>
   </strongier>
   <threshold> 0 </threshold>
</Detector>
```

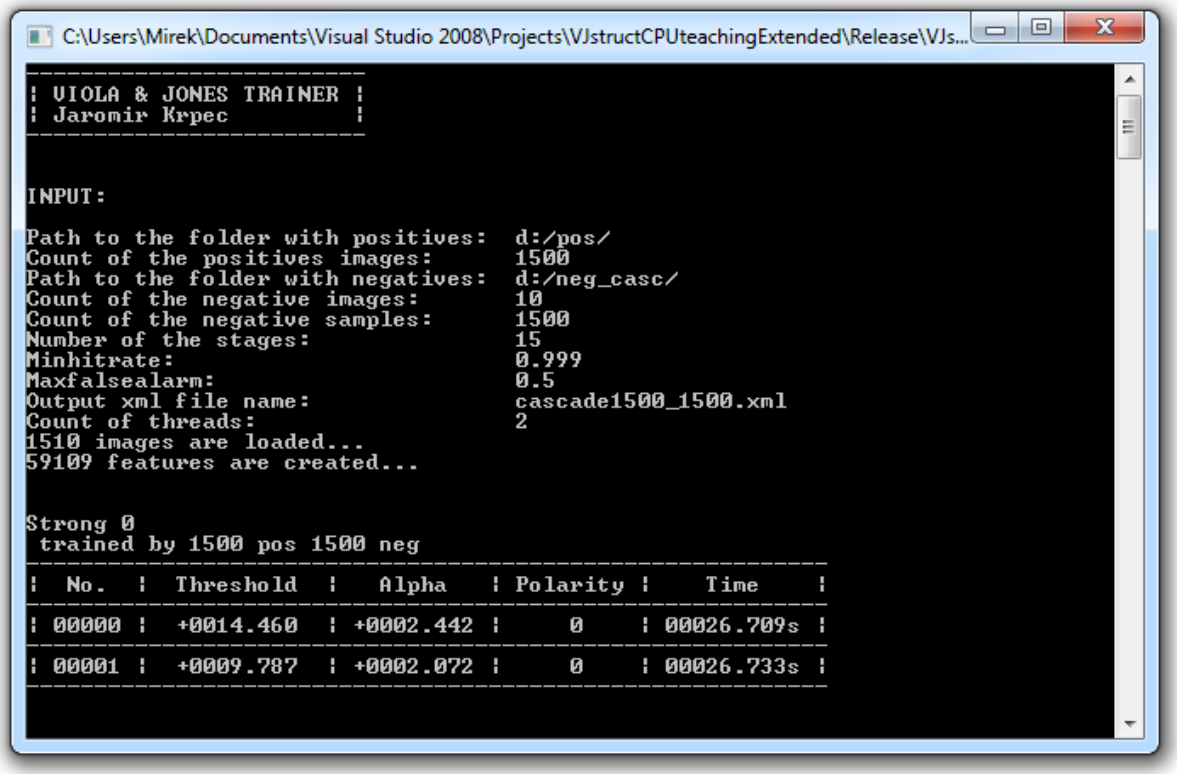
E. Program pro učení kaskády

Pro správnou funkčnost programu pro trénování množiny klasifikátoru je třeba zadat následující parametry:

- Cesta k pozitivní množině obrázků ve formátu PGM
- Počet obrázků, které se mají použít
- Cesta k negativní množině obrázků složených z negativních vzorů ve formátu PGM
- Počet negativních složených obrázků
- Počet negativních obrázků vybraných pro trénování stupně kaskády
- Velikost kaskády (počet silných klasifikátorů)
- Minimální požadovaný detection rate pro každý stupeň kaskády
- Maximální požadovaný false alarm pro každý stupeň kaskády
- Jméno výstupního XML souboru s natrénovanými klasifikátory
- Počet vláken, která budou přidělena k učení klasifikátorů

Požadavky pro spuštění:

- OS Microsoft Windows 7 64bit



F. Program pro detekci

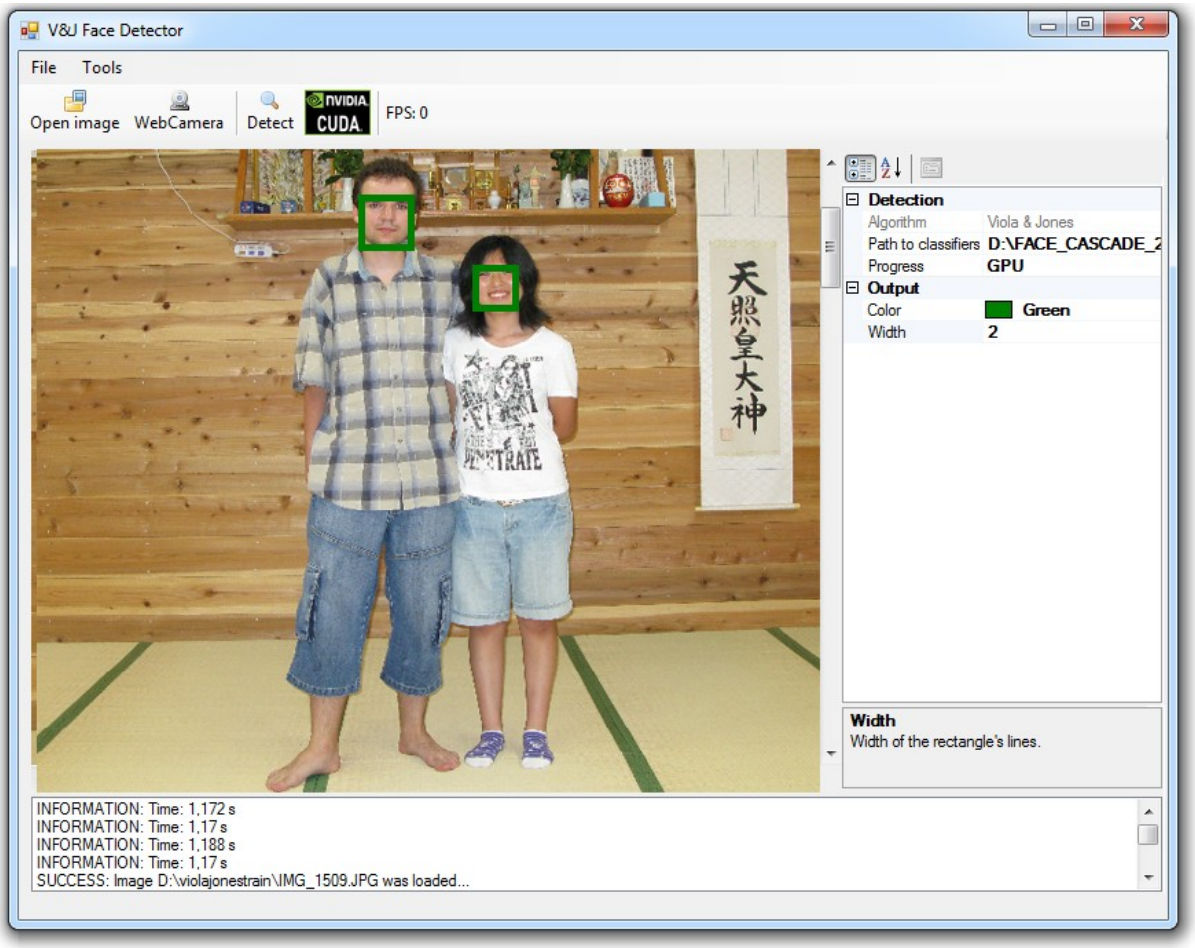
Program pro detekci umožňuje zjišťovat, zda se obrazy nachází v obrazech běžných grafických formátů, ale zároveň je možné zpracovávat i vstup z webkamery připojené k PC. Nalezené obličeje se zaznamenávají přímo v grafickém výstupu aplikace. Navíc jsou vypisovány informace o délce detekce na aktuálním vstupu. V případě vstupu z webkamery je zjišťováno FPS (frames per second), čili počet obrazů zpracovaných za jednu sekundu.

V programu pro detekci obličejů je možné nastavit:

- Provedení detekce pomocí CPU nebo GPU
- Vstupní XML soubor s natrénovanými klasifikátory
- Styl ohraničení nalezených obličejů (tloušťka čáry a barva)

Požadavky pro spuštění:

- OS Microsoft Windows 7 64bit
- .NET Framework 3.5
- V případě detekce pomocí GPU, grafická karta podporující technologii CUDA a nainstalovaný NVIDIA CUDA 4.1.1 driver
- Program byl testován na počítačových sestavách s grafickými kartami
  - GeForce GTX560
  - GeForce GTX560Ti
  - Quadro600



G. Obsah přiloženého DVD

Na DVD jsou umístěny tyto adresáře:

- Zdrojové kódy pro Microsoft Visual Studio 2008
  - `\source_codes\`
  - `\source_codes\EmguCV_frontalface\`
    - `Zdrojový kód detektoru využívajícího plně OpenCV`
  - `\sources_codes\ViolaJonesExtended\`
    - zdrojový kód učitele
  - `\sources_codes\VJ_Detector_GUI\`
    - zdrojový kód spustitelné aplikace pro detekci
  - `\source_codes\FaceDetectCPU\`
    - zdrojové kódy knihovny pro CPU detekci
  - `\source_codes\FaceDetectGPU\`
    - zdrojové kódy knihovny pro GPU detekci

- XML s klasifikátory
  - `\classifiers\`

- Spustitelný program se všemi potřebnými knihovnami
  - `\applications\`
  - `\applications\teacher\`
  - `\applications\detector\`
  - `\applications\cv\`

- Množina složených obrazů neobsahující obličej a vzorky obsahující obličej pro testování kaskády
  - `\train_images\`
  - `\train_images\negatives\`
  - `\train_images\positives\`

- Množina obrázků pro ověření funkčnosti detektoru
  - `\test_images\`

- Text diplomové práce
  - `\text\`